VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

KLASIFIKACE OBRAZŮ POMOCÍ GENETICKÉHO PROGRAMOVÁNÍ

IMAGE CLASSIFICATION USING GENETIC PROGRAMMING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. KAROLÍNA JAŠÍČKOVÁ**
AUTHOR

VEDOUCÍ PRÁCE **Prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Jašíčková Karolína, Bc.**
Obor: Bioinformatika a biocomputing
Téma: **Klasifikace obrazů pomocí genetického programování**
Image Classification Using Genetic Programming
Kategorie: Umělá inteligence

Pokyny:

1. Seznamte se s problematikou genetického programování a jeho využitím pro klasifikaci obrazů.
2. Navrhněte metodu založenou na genetickém programování pro klasifikaci obrazů. Zaměřte se na generování takových klasifikátorů, kterou jsou jednoduché a současně účinné.
3. Vytvořenou metodu implementujte a experimentálně ověřte na zadaných testovacích sadách.
4. Porovnejte dosažené výsledky s jinými přístupy.
5. Zhodnoťte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Sekanina Lukáš, prof. Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Tato práce se zabývá klasifikací obrazu pomocí genetického programování a koevoluce. Algoritmy genetického programování umožňují generovat spustitelné struktury a navrhovat tak automatizovaně řešení ve formě programů. Použití koevoluce s predikcí fitness snižuje časovou náročnost výpočtu fitness a tím i dobu trvání celého algoritmu. Práce popisuje teoretický základ evolučních algoritmů a zejména kartézské genetické programování. Jsou také popsány vlastnosti koevolučních algoritmů a zejména navržená metoda pro návrh klasifikátoru obrazu s využitím koevoluce fitness prediktorů, jejímž cílem je nalézt kompromis mezi přesností klasifikace, dobou návrhu a složitostí klasifikátoru. Součástí práce je implementace navžené metody, provedení experimentů a srovnání získaných výsledků s ostatními metodami.

Abstract

This thesis deals with image classification based on genetic programming and coevolution. Genetic programming algorithms make generating executable structures possible, which allows us to design solutions in form of programs. Using coevolution with the fitness prediction lowers the amount of time consumed by fitness evaluation and, therefore, also the execution time. The thesis describes a theoretical background of evolutionary algorithms and, in particular, cartesian genetic programming. We also describe coevolutionary algorithms properties and especially the proposed method for the image classifier evolution using coevolution of fitness predictors, where the objective is to find a good compromise between the classification accuracy, design time and classifier complexity. A part of the thesis is implementation of the proposed method, conducting the experiments and comparison of obtained results with other methods.

Klíčová slova

evoluční algoritmy, genetické programování, kartézské genetické programování, koevoluce, aproximace fitness, fitness prediktory, klasifikace obrazu

Keywords

evolutionary algorithms, genetic programming, cartesian genetic programming, coevolution, fitness approximation, fitness predictors, image classification

Citace

JAŠÍČKOVÁ, Karolína. *Klasifikace obrazů pomocí genetického programování*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Prof. Ing. Lukáš Sekanina, Ph.D.

Klasifikace obrazů pomocí genetického programování

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením pana prof. Ing. Lukáše Sekaniny, Ph.D. Uvedla jsem všechny literární prameny, ze kterých jsem čerpala.

.....
Karolína Jašíčková
18. května 2018

Poděkování

Ráda bych poděkovala prof. Ing. Lukáši Sekaninovi, Ph.D. za odborné vedení, cenné rady a konzultace k vypracování této diplomové práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Existuje velké množství úloh, které jsou pouze obtížně řešitelné pomocí konvenčních přístupů[1] z důvodu vysoké časové složitosti, nebo na nich tyto přístupy přímo selhávají. Díky této situaci se do popředí stále více dostává umělá inteligence a s ní i oblast natural computingu. Příroda se stala velkou inspirací pro návrh metod, které dokážou řešit zejména takové úlohy, jejichž řešení je obtížně specifikovatelné a kde existuje vysoký stupeň neurčitosti. Od počátku výzkumu v této oblasti vznikla řada dodnes používaných metod jako např. neuronové sítě, evoluční algoritmy, celulární automaty nebo metody inspirované inteligencí hejna. Konkrétně algoritmy inspirované přírodním procesem evoluce přinesly v mnoha oblastech nové možnosti s velmi slibnými výsledky [11]. Tato práce používá právě evoluční algoritmy, a to zejména pro jejich vysokou schopnost optimalizace řešení.

Úlohou řešenou v této práci je klasifikace obrazu. Obecně je v oblasti klasifikace obrazu snaha se co nejvíce přiblížit lidské schopnosti rozpoznávat jednotlivé objekty ze všech možných úhlů. Problémem je však obrovské množství možných objektů a způsobů jejich zachycení v obraze. Z těchto důvodů se jedná o velmi složitou úlohu. Zatím nejúspěšnějším a nejpoužívanějším přístupem pro řešení tohoto nesnadného úkolu je použití hlubokých neuronových sítí [9]. Jejich použití s sebou však nese i nevýhody. Je nutné mít k dispozici poměrně velké množství výpočetních zdrojů, což omezuje možnosti využití např. v nízkopříkonových hardwarových aplikacích. Dnešním trendem je minimalizace rozměrů zařízení a jejich spotřeby. Nasazení hlubokých neuronových sítí se proto v této oblasti začíná prosazovat jen pozvolna. Naopak úspěšně se pro návrh hardware používá kartézské genetické programování (CGP). Cílem této práce je pomocí CGP navrhnout klasifikátor specializovaný na rozpoznávání ručně psaných číslic, který bude pro danou úlohu spolehlivý a zároveň bude mít vlastnosti výhodné pro použití v nízkopříkonovém hardware. Protože největší nevýhodou evolučních algoritmů je jejich potřeba značného množství výpočetního času na jeden běh, je v této práci CGP kombinováno s aproximací fitness pomocí fitness prediktoru v rámci koevolučního algoritmu. Výsledná koevoluční metoda by měla přinést oproti základní variantě CGP značné urychlení návrhu klasifikátoru.

V kapitole 2 jsou popsány obecné principy evolučních algoritmů, na kterých je postavena i použitá metoda. Je zde nastíněno dělení evolučních algoritmů, jsou probrány možné způsoby kódování chromozomu a genetické operátory. Speciální kapitola je také věnována genetickému programování. CGP a jeho vlastnosti jsou posány v kapitole 3. Koevoluční algoritmy a jejich použití pro aproximaci fitness, spolu s výhodami a nevýhodami těchto algoritmů, je nastíněno v kapitole 4. Kapitola 5 pak popisuje způsob návrh klasifikátoru,

[1] Konvenčním přístupem rozumíme techniky a postupy běžně používané v IT

použití prediktorů pro CGP a vlastní algoritmus metody. Nastavení experimentů v kapitole 6 popisuje použitou datovou sadu a nastavení parametrů algoritmu, se kterým byly experimenty provedeny. Následně v kapitole 7 jsou prezentovány výsledky dosažené s popsaným nastavením. Tyto výsledky jsou také porovnány s dalšími přístupy používanými pro klasifikaci obrazu. Kapitola 8 popisuje možné modifikace použité metody, které by měly vést k lepším výsledkům, než kterých bylo v této práci dosaženo. V závěrečné kapitole 9 je shrnuta odvedená práce a provedeno zhodnocení výsledků experimentů.

Kapitola 2

Evoluční algoritmy

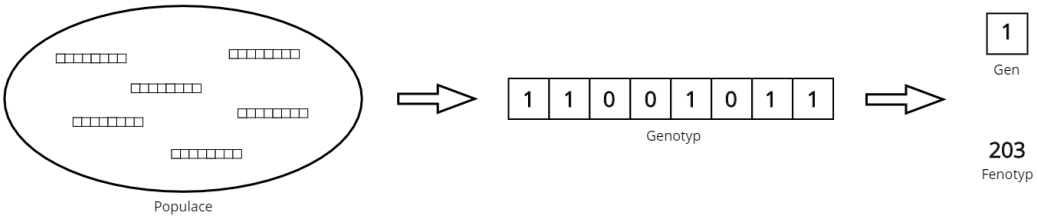
Třída algoritmů inspirovaná biologickým mechanismem evoluce se nazývá evoluční algoritmy (EA). Jedná se o metody prohledávání prostoru, jejichž cílem je naléz optimální řešení zadané úlohy. V porovnání s konvenčními metodami se jejich aplikace na určité typy problémů se ukázala jako velmi účinná. Používají vhodnou reprezentaci problému pro zakódování řešení do chromozomu. Tímto způsobem jsou vytvoření jedinci, kteří jsou pomocí geneticky inspirovaných operátorů selekce, křížení a mutace modifikováni za účelem nalezení nejlepšího možného řešení. K určení a porovnání kvality jedinců se používají různé fitness funkce. Díky evoluci se jedinci postupně přizpůsobují svému prostředí tvořenému zadanou úlohou. Ti s lepší fitness hodnotou než zbytek populace mají vysokou šanci, že budou vybráni do fáze reprodukce, zatímco slabší jedinci jsou s velkou pravděpodobností vyřazováni z populace. Přeživší jedinci jsou schopni plodit potomky, tj. nová kandidátní řešení, a předat dál své geny. Pojmy používané v EA, jejich dělení, jimi používané operátory a kódování a další principy jsou podrobněji popsány v následujících podkapitolách, které jsou zpracovány podle [12] [10] a [5].

2.1 Pojmy

Evoluční algoritmy používají pojmy adoptované z biologie. Jelikož jsou používány v celém následujícím textu, je vhodné je definovat.

- **Gen** – základní element chromozomu a nositel informace. Nabývá jedné hodnoty z konečné množiny nad definovanou abecedou.
- **Chromozom, jedinec, genotyp** – soubor genů. Reprezentuje jeden stav z prohledávaného stavového prostoru.
- **Fenotyp, kandidátní řešení** – potenciální řešení zadaného problému. Sestavuje se na základě genotypu, resp. se jím daný genotyp projevuje.
- **Populace** – struktura obsahující n chromozomů ($n \geq 1$) s danou reprezentací. Z matematického pohledu se jedná o multimnožinu.
- **Fitness** – udává kvalitu kandidátního řešení.
- **Fitness funkce** – přiřazuje kandidátnímu řešení fitness hodnotu.
- **Evaluace** – výběr fitness hodnoty jedince, neboli jeho ohodnocení.

- **Selekce** – proces výběru jedinců k reprodukci na základě fitness hodnot. Obvykle obsahuje i stochastický prvek.
- **Křížení, mutace** – variační operátory sloužící k tvorbě potomků z rodičů.
- **Generace** – jedna iterace EA, v rámci které dochází ke změnám v kandidátních řešeních (evoluci). Z populace P_t vznikne nová populace P_{t+1}, $t \geq 0$, aplikací mechanismu selekce, variačních operátorů a mechanismu nahrazení původních jedinců.



Obrázek 2.1: Vztah mezi genem, jedincem, fenotypem a populací

2.2 Dělení evolučních algoritmů

Mezi jednotlivými EA lze najít řadu rozdílů i podobností. Na základě těchto vlastností je se obvykle dělí do následujících kategorií:

- genetické algoritmy,
- evoluční strategie,
- evoluční programování,
- genetické programování a
- ostatní.

Genetické algoritmy (GA) kladou důraz na souběžný vliv selekce, křížení a mutace na genotyp (viz kap. 2.7). Tyto operátory mají pravděpodobnostní charakter. Pro zakódování problému používají GA řetězce bitů, celých čísel, či jiných elementů. Typická velikost populace je v desítkách až stovkách jedinců.

Klasická varianta evoluční strategie (ES) se oproti ostatním EA vymezuje několika vlastnostmi. Kandidátní řešení jsou zakódována pomocí vektoru reálných čísel. Selekce pro výběr rodičů je nezaujatá, tedy probíhá náhodně s uniformním rozložením. Selekce do nové populace je ordinální a deterministická, což znamená, že vždy přežívá daný počet nejlepších jedinců z populací rodičů a potomků. Všechny genetické operátory jsou parametrizované, a tudíž je lze řídit. Jedinci nesestávají pouze z kandidátního řešení, ale také řídicích parametrů (např. míra mutace). ES se dělí do podkategorií podle celkového počtu rodičů v generaci μ, počtu potomků λ, počtu partnerů podílejících se na jednom potomkovi ρ a maximálního věku jedince κ. Základní dělení lze vidět v tabulce 2.1.

Evoluční programování je podobné evolučním strategiím. Pro tvorbu nových jedinců používá pouze mutaci, jejíž četnost je součástí jedince a se zvyšující se fitness klesá. Nepoužívá žádnou jednotnou reprezentaci kandidátních řešení, namísto toho je jeho volba závislá čistě na řešené úloze a může mít prakticky libovolnou podobu.

Algoritmus	Rodiče	Partneři	Potomci	Věk
$(1+1)$	$\mu = 1$	$\rho = 1$	$\lambda = 1$	$\kappa = \infty$
$(1+\lambda)$	$\mu = 1$	$\rho = 1$	$\lambda \geq 1$	$\kappa = \infty$
$(1,\ \lambda)$	$\mu = 1$	$\rho = 1$	$\lambda \geq 2$	$\kappa = 1$
$(\mu+1)$	$\mu \geq 2$	$\rho = 2$	$\lambda = 1$	$\kappa = \infty$
$(\mu+\lambda)$	$\mu \geq 2$	$\rho = 2$	$\lambda \geq 2$	$\kappa = \infty$
$(\mu,\ \lambda)$	$\mu \geq 2$	$\rho = 2$	$\lambda \geq \mu$	$\kappa = 1$

Tabulka 2.1: Dělení evolučních strategií dle [10]

Genetické programování (GP) navrhuje spustitelné struktury, které jsou reprezentovány grafem, nejčastěji stromem nebo acyklických orientovaným grafem. Podrobněji než ostatní kategorie evolučních algoritmů je GP popsáno v sekci 2.7. EA však často kombinují různé přístupy, a proto je lze jen zřídka jasně zařadit do jedné kategorie.

2.3 Kódování chromozomu

Schopnost zakódovat možná řešení úlohy je nutná pro použití evolučních algoritmů. Kódování je do jisté míry závislé na konkrétní řešené úloze a jeho návrhem začíná řešení úlohy. Nevhodné kódování může způsobit celou řadu problémů včetně neuspokojivých výsledků algoritmu. Mezi standardní kódování patří

- binární,
- permutační,
- číselné a
- stromové.

Binární kódování je nejčastější formou, kdy je chromozom vyjádřen přímo jako binární řetězec. Může kódovat celočíselnou či reálnou hodnotu z libovolného konečného rozsahu hodnot. To ovšem není ideální, protože velikost změny hodnoty je značně závislá na pozici změněného bitu. Pro malé změny může být nutné invertovat bitů více a naopak změna jednoho bitu může zásadně změnit fenotyp a následně i fitness. Z tohoto důvodu se pro kódování čísel používají speciální kódy jako je Grayův kód. Toto řešení však vyžaduje režii navíc a neřeší problémy zcela. Naopak zcela ideální je toto kódování pro problémy jako knapsack[1].

Pro permutační problémy typu problém obchodního cestujícího se nejčastěji používá permutační kódování. V chromozomu se vyskytují navzájem odlišné znaky či hodnoty odpovídající řazeným položkám. Řetězec čtený zleva doprava pak odpovídá zakódovanému pořadí ve formě permutace. Pro tento typ kódování je nutné používání speciálních genetických operátorů pro zachování permutace v řetězci.

Některé úlohy nelze zakódovat jedinou hodnotou, jak je tomu v případě binárního kódování. V takových případech se jako chromozomy používají řetězce celých a reálných čísel, znaků či komplikovanějších objektů, které zároveň řeší problémy binárních chromozomů. Takto reprezentovaná může být například cesta bludištěm, pohyb po šachovnici, barvení

[1] U tohoto problému jsou dány položky s definovanou hodnotou a velikostí. Dále je dán batoh (knapsack) určité kapacity a cílem je maximalizovat hodnotu věcí, které jsou do něj vloženy, ale nepřekročit jeho kapacitu. Pro tuto úlohu jednotlivé bity v chromozomu odpovídají příslušným položkám ze seznamu. Hodnota každého bitu pak značí, jestli je položka v batohu nebo ne.

grafu různými barvami a další. Nevýhodou tohoto kódování je, že pro některé problémy může být potřeba použít nové typy genetických operátorů.

Kódování do stromu je typické pro genetické programování, které vyvíjí programy, výrazy a jiné spustitelné struktury. Standardně se jedná o binární strom a nachází se v něm objekty jako funkce a příkazy. Chromozom nemusí mít konstantní délku, což může vést k problémům jako je generování částí stromů bez přínosu ke kvalitě řešení za stagnace fitness, protože jsou vkládány neutrální výrazy. Tyto problémy se dají řešit například omezením hloubky stromu. Klasickou úlohou, která používá tuto reprezentaci, je symbolická regrese.

2.4 Fitness funkce

Fitness funkce je funkce, která bere kandidátní řešení jako vstup a produkuje numerický výstup, který říká, jak kvalitní toto řešení je. Výpočet fitness hodnoty se v průběhu evoluce provádí opakovaně v každé generaci pro každého jedince. Z tohoto důvodu by její samotný výpočet měl být co nejrychlejší, aby algoritmus příliš nezpomaloval. Vypočítané fitness hodnoty se v průběhu evoluce používají pro selekci a tvorbu nové populace. Fitness funkce umožňuje vytvářet selekční tlak, který pomáhá konvergenci populace k hledanému řešení. Cílem evolučního algoritmu je nalézt globální extrém fitness funkce. Podle typu hledaného extrému lze úlohy dělit na maximalizační a minimalizační. Pokud se podaří globální extrém najít, tak se podařilo najít perfektní řešení dané úlohy. Někdy se může stát, že populace ustrne v lokálním extrému funkce. Tento jev je pozorovatelný zejména u multimodálních funkcí, pro které existují speciální techniky, které se snaží pomocí řízení populace lokální extrém překonat. U některých úloh hodnota globálního extrému není předem známá nebo nemusí být možné se do něj dostat, a pak hledáme co nejlepší přijatelné řešení.

2.5 Genetické operátory

Aby bylo možné vytvářet nové generace jedinců ze stávající populace a umožnit tak evoluci, je potřeba mít definované genetické operátory. Evoluční algoritmy definují tři základní operátory, které společným použitím umožňují generovat nové jedince (potomky) z rodičovských chromozomů. Konkrétně jsou to selekce pro výběr rodičů, křížení a mutace. Jejich konkrétní implementace je závislá na použitém kódování chromozomu.

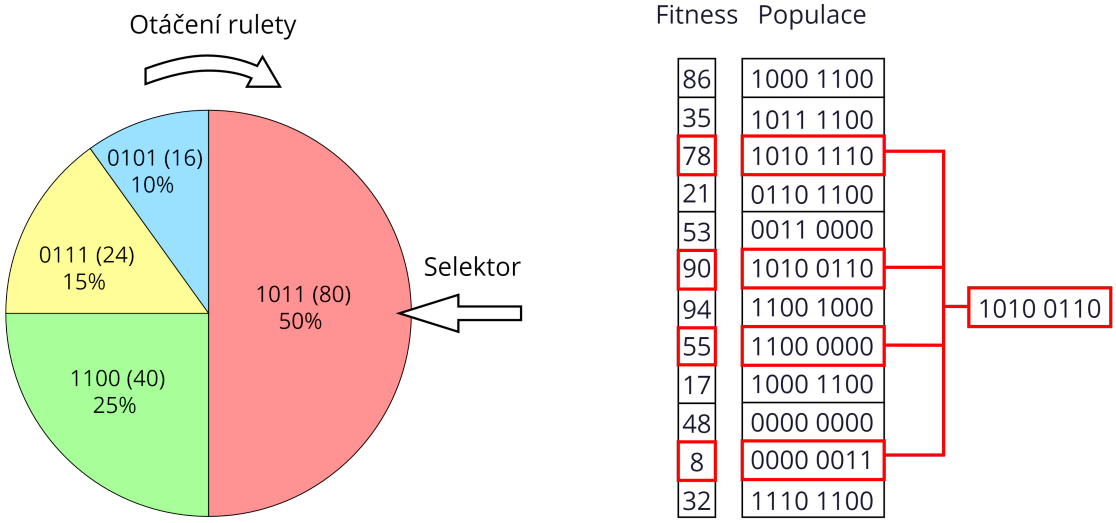
2.5.1 Selekce

Pro vznik nových potomků je potřeba mít rodiče. Platí, že větší šanci se reprodukovat mají silnější jedinci. Slabší jedinci mají šanci nižší, ale stále mohou své geny předat dále. Tímto způsobem je vytvářen selekční tlak a zároveň zachována důležitá genetická rozmanitost. Pokud by geny předávaly jen nejsilnější jedinci, začalo by postupně docházet k degeneraci populace, či její stagnaci.

V evolučních algoritmech je používáno několik způsobů selekce. Na obrázku 2.2 jsou znázorněny dva základní – ruleta a turnaj. Při použití ruletové metody je nejprve fitness všech jedinců přepočítána na pravděpodobnost výběru. Tento výpočet se provede jako

$$P_i = \frac{f_i}{\sum_{j=1}^{n} f_j}, \tag{2.1}$$

kde P_i je pravděpodobnost výběru jedince i, f_i jeho fitness hodnota, f_j je fitness j-tého jedince z populace o velikosti n. Po získání těchto pravděpodobností dochází k vlastní selekci. Princip lze vysvětlit na příkladu točení ruletou, na které každému jedinci náleží plocha úměrně velká jeho pravděpodobnosti výběru. Jedinec, na kterého po zastavení ukazuje selektor, je vybrán jako rodič. Točení ruletou se provádí opakovaně pro potřebný počet rodičů. Existuje i varianta rulety, kde je selektorů tolik, kolik je vybíraných rodičů, a jsou rovnoměrně rozmístěny po obvodu rulety. V každé generaci se pak všichni rodiče vyberou jediným zatočením.



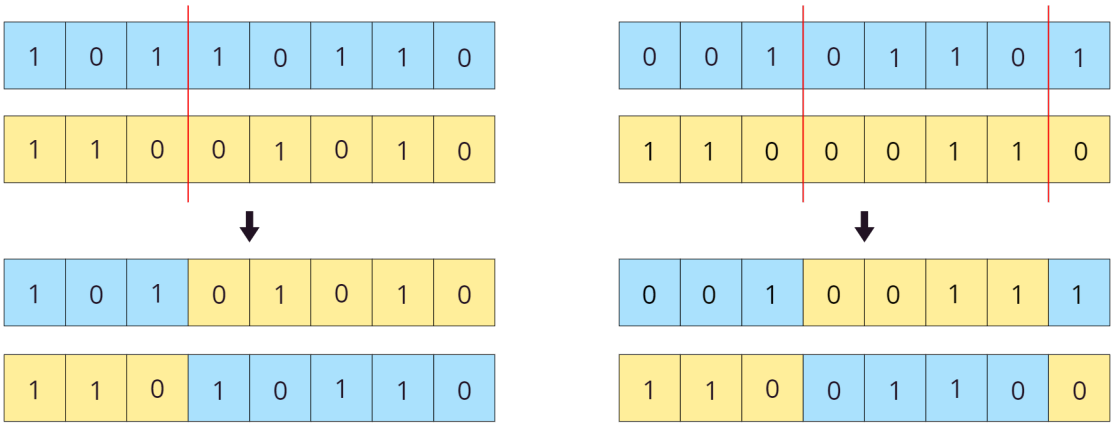
Obrázek 2.2: Selekce a) ruletovou metodou (vlevo), b) turnajovou metodou s velkostí turnaje $k = 4$ (vpravo)

Turnajová metoda nejprve z populace náhodně vybere předem daný počet jedinců (tzv. velikost turnaje k). Fitness hodnoty vybraných jedinců jsou následně vzájemně porovnány a jedinec s nejlepší fitness se stává vítězem, resp. jedním z vybraných rodičů. Stejně jako u ruletové metody je tento postup opakován, dokud nejsou získáni všichni rodiče.

2.5.2 Křížení

Křížení kombinuje chromozomy rodičů k vytvoření nových jedinců. Popíšeme čtyři základní metody, kterými lze křížení provádět. První a nejjednodušší z nich je jednobodové křížení. Nejprve je náhodně stanoven bod křížení, který udává místo prohození konců rodičovských chromozomů k následnému získání potomků. Vícebodové křížení je druhým používaným přístupem. Na rozdíl od jednobodového křížení nepoužívá jeden, ale více bodů rekombinace. Ostatní mechanismy, jako je způsob výběru bodů křížení či způsob skládání chromozomů potomků, jsou stejné jako u jednobodového křížení. Jednobodové a dvoubodové křížení je ilustrováno na obrázku 2.3 Třetí metodou je uniformní křížení. V tomto případě nejsou prohazovány části chromozomů jako u předchozích metod. Na místo toho je uvažován každý gen zvlášť. Je stanovena pravděpodobnost, se kterou budou geny brány od jednoho či druhého rodiče. Obvykle je její hodnota $0,5$, ale lze pro každého rodiče nastavit jinou pravděpodobnost. Poslední metodou je aritmetické křížení, které lze aplikovat na chromozomy kódující číselnou hodnotu. Namísto použití jednotlivých částí chromozomů

je použita aritmetická operace nad rodiči, kterou je typicky vážený průměr. Pro každého potomka lze nastavit jinou váhu. Kromě těchto čtyř metod existuje i řada metod pro permutační problémy, protože klasické křížení nezachovává permutaci. Zajímavou možností je také použití tří a více rodičů namísto dvou.

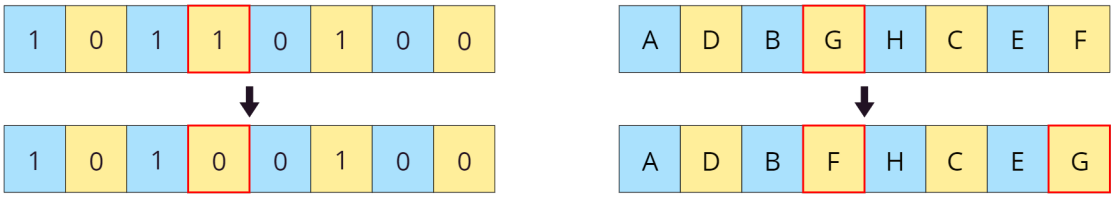


Obrázek 2.3: Křížení a) jednobodové (vlevo), b) dvoubodové (vpravo)

2.5.3 Mutace

V biologii je mutace trvalá změna nukleotidové sekvence genomu, která se může projevit také změnou fenotypu. V evolučních algoritmech funguje mutace stejným způsobem. Jedná se o změnu jedné či více hodnot v chromozomu jedince, která se může projevit na hodnotě jeho fitness. Typicky se mutace provádí až po selekci a křížení.

Některé algoritmy mohou provádět mutaci jednoho či více genů s určitou zadanou pravděpodobností, jiné ji mohou vykonávat při vygenerování každého nového jedince. Samotný počet mutovaných genů se také může měnit a lze jej zadat jako vstupní parametr algoritmu. Jinak je také mutace implementována např. pro binárně kódované a permutační problémy. U binárního kódovaní je dostačující pouze invertovat hodnotu daného počtu bitů. Permutační kódování si s sebou nese při mutování stejné problémy jako při křížení. Je tedy opět nutné buďto navrhnout takový mutační operátor, který bude permutaci zachovávat, nebo zmutovaný chromozom dodatečně opravit. Příklady mutace na těchto dvou typech chromozomů jsou znázorněny na obrázku 2.4. V případě permutačního chromozomu je pak vidět, že se mutací změní nikoli jeden, ale dva geny. Původní a výsledná alela mutovaného genu udává, které dva geny byly prohozeny, aby byla permutace zachována.



Obrázek 2.4: Mutace na a) binárním (vlevo), b) permutačním chromozomu (vpravo)

2.6 Obecný algoritmus

Všechny EA sdílí stejný základní algoritmus. Poloformálně zapsán vypadá následovně:

1. Náhodně vygeneruj počáteční populaci velikosti M
2. Ohodnoť všechny jedince
3. Dokud není splněna některá ukončující podmínka:
   3.1. Vyber rodiče pro tvorbu potomků
   3.2. Vytvoř nové jedince
   3.3. Ohodnoť nové jedince
   3.4. Vytvoř novou populaci
4. Jedince s nejlepší fitness vrať jako řešení

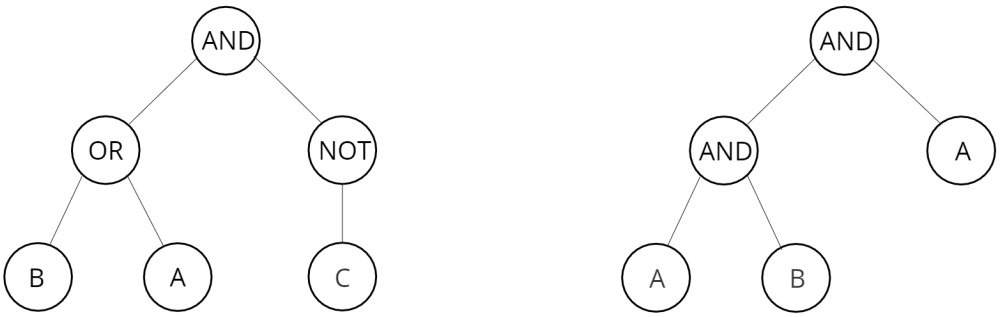
Na počátku algoritmu je vždy nejprve náhodně inicializována a ohodnocena počáteční populace. V některých případech je možné počáteční populaci inicializovat vhodnou heuristikou nebo známým řešením. Její velikost M je vstupním parametrem algoritmu a běžně se již v průběhu evoluce nemění. S počáteční populací je následně zahájena evoluce. Ze stávající populace jsou některou selekční metodou vybráni rodiče a je aplikováno křížení a mutace k vytvoření nových jedinců. Následuje vytvoření nové populace, která je tvořena poměrným zastoupením rodičů a jejich potomků. Tento poměr závisí na konkrétním algoritmu. Je možné také aplikovat elitismus, při kterém se do nové populace vždy dostává nejlepší jedinec z populace předchozí. Tento postup se aplikuje, dokud není splněna ukončující podmínka algoritmu. Typicky se jedná o vyčerpání dovoleného počtu generací nebo dosažení nejlepší možné fitness hodnoty.

2.7 Genetické programování

Skupinou evolučních algoritmů, které umožňují vytvářet programy, je genetické programování. V jistém smyslu je snahou pouze specifikovat požadovaný výstup programu, ale už ne způsob jeho dosažení a nechat počítač vytvořit kód implementace. Stejně jako ostatní evoluční algoritmy pracuje GP s definicí cíle pomocí fitness, která je používána k vývoji populace kandidátních řešení. Používány jsou i stejné genetické operátory. Na rozdíl od ostatních tříd evolučních algoritmů pro ně však není přirozené používat jako chromozom řetězec fixní délky. Programy jsou ve své podstatě hierarchické, mají proměnnou délku a obsahují v obecné podobě iterace a rekurzi. Kódování pro genetické programování musí tyto vlastnosti odrážet. Obvyklá je reprezentace syntaktickým stromem, ale existují i lineární či jiné grafové reprezentace. Stromy jsou stavěny rekurzivně z množiny funkcí a množiny terminálů definovaných pro řešenou úlohu. Všechny funkce přijímají pevný počet argumentů. Může se jednat o aritmetické operace, booleovské operace, podmínky, cykly a jiné pro řešený problém specifické funkce.

Vzhledem k reprezentaci chromozomů je potřeba mít speciální metody na inicializaci populace. Počáteční jedinci jsou vytváření kombinací funkcí a terminálů z definovaných množin. Nechť T je množina terminálů a F je množina funkcí. V případě GP založeného na stromech je na začátku evoluce stanovena maximální hloubka stromu v počáteční populaci. Jednou z používaných metod je úplná metoda (tzv. full), u které se všechny listy vytvořeného stromu nachází ve stejné hloubce. Všechny počáteční stromy však nemusí mít

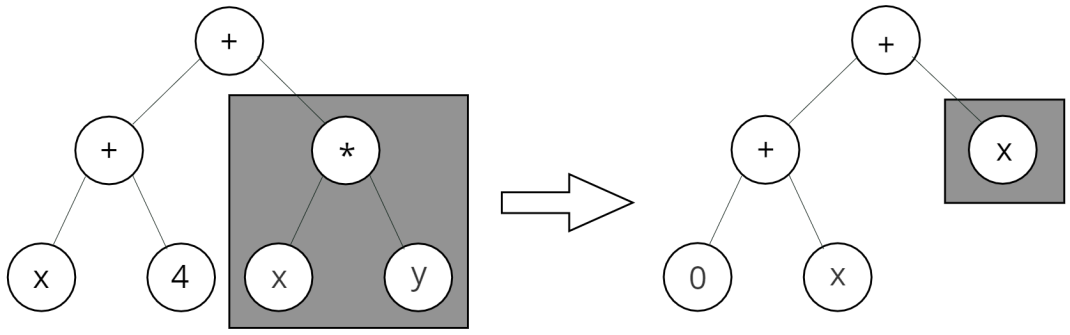
stejný tvar a počet uzlů. Tyto vlastnosti jsou ovlivněny aritou funkcí v nelistových uzlech. Vytváření stromu úplnou metodou probíhá tak, že v nelistových uzlech jsou vybírány pouze funkce a v maximální hloubce poté jen terminály. O něco složitější je růstová metoda (tzv. grow). Nejprve se náhodně vybere některá z funkcí jako kořen stromu. Další uzly jsou náhodně vybírány z množiny funkcí i terminálů, až strom dosáhne maximální hloubky, ve které jsou opět pouze terminály. Pokud je v některé větvi vybrán terminál, růst v ní dále nepokračuje a uzel se stává listovým. Tyto dvě metody lze i kombinovat tak, že se každá použije na vygenerování části populace. Zároveň je možné volit pro skupiny jedinců jinou maximální hloubku stromu. Příklady stromů vygenerovaných těmito dvěma metodami jsou na obrázku 2.5.



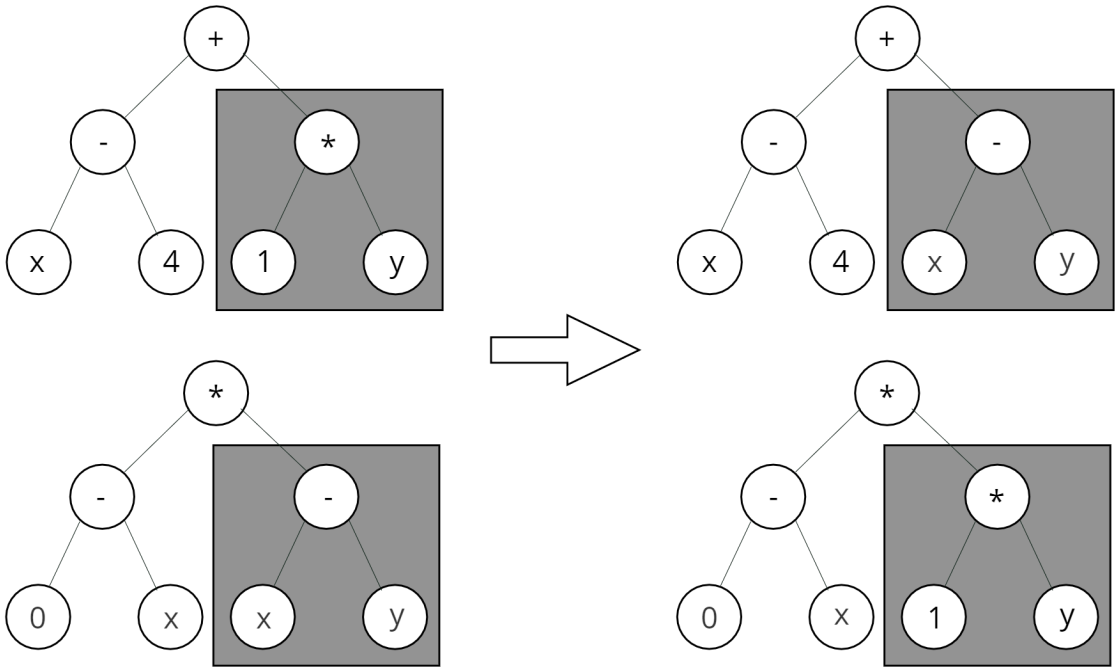
Obrázek 2.5: Inicializace stromového chromozomu a) úplnou (vlevo), b) růstovou (vpravo) metodou pro $F = \{AND, OR, NOT\}$ a $T = \{A, B, C\}$.

Fitness hodnota jedinců je počítána pomocí spouštění vyvinutých programů a značí, do jaké míry program plní úlohu. Často je tato hodnota počítána na sadě fitness případů (tzv. fitness cases). Každý fitness případ představuje situaci, na které může být hodnocena činnost programu. Příkladem může být symbolická regrese zadaná pomocí několika bodů, kdy každý bod je jedním fitness případem. Pro každý bod je vytvořeným programem vypočítána hodnota představující odchylku od očekávané hodnoty. Takto získané hodnoty se sčítají a tvoří výslednou fitness hodnotu kandidátního programu.

Jak již bylo zmíněno, genetické operátory jsou stejné nebo alespoň velmi podobné jako u ostatních evolučních algoritmů. V selekci genetické programování nezavádí žádné změny. Odlišnosti při křížení a mutaci jsou dány pouze jinou reprezentací chromozomu, ale princip zůstává zachován. Při křížení je nejprve v rodičovském chromozomu náhodně vybrán uzel, ve kterém bude bod křížení. Podstrom s tímto kořenovým uzlem je fragmentem křížení pro daného rodiče. Rodiče si fragmenty navzájem vymění a vzniknou tak potomci. Mutace probíhá velmi podobně. Popsána bude jedna z jejích častých variant. V daném jedinci je nejprve vybrán uzel jako bod křížení a jím daný podstrom je odstraněn. Na jeho místo se vygeneruje nový náhodný podstrom. Tyto dvě operace jsou ilustrovány na obrázcích 2.6 a 2.7. Šedě vyznačené části stromů jsou fragmenty, které se mezi rodiči vyměňují, nebo se mutují. Ostatní vlastnosti genetického programování jako obecný algoritmus, základní parametry evoluce, atd. zůstávají stejné jako u ostatních evolučních algoritmů. U genetického programování využívajícího lineární kódování chromozomu se genetické operátory křížení a mutace aplikují stejným způsobem, jak bylo popsáno v předchozích podkapitolách 2.5.2 a 2.5.3.



Obrázek 2.6: Mutace na stromovém chromozomu



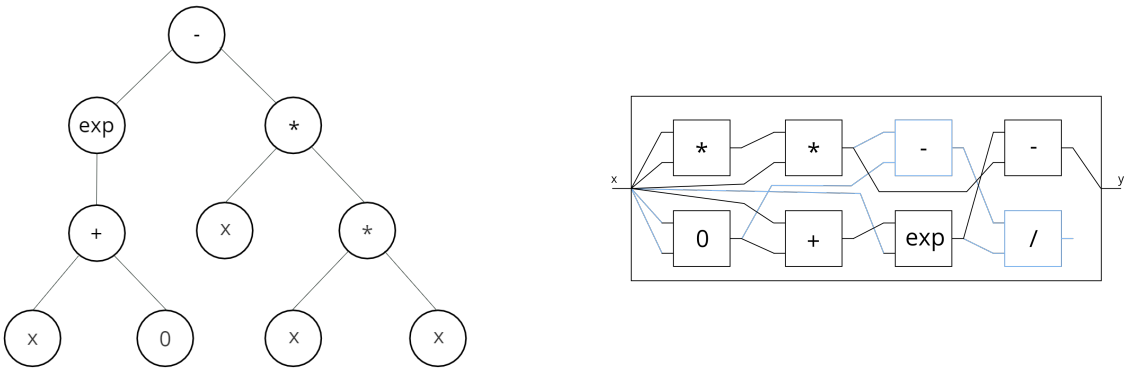
Obrázek 2.7: Křížení na stromovém chromozomu. Rodiče jsou vlevo a potomci vpravo.

Typickou úlohou pro genetické programování je symbolická regrese. Jedná se o typ regresní analýzy, která prohledává prostor matematických výrazů za účelem nalezení modelu, který nejlépe odpovídá zadané datové sadě. Datová sada typicky obsahuje až desetitisíce experimentálně získaných hodnot. Předpokládá se, že výstupní hodnota závisí na vstupní, a je tedy možné tuto závislost vyjádřit funkcí. Není vyžadován žádný počáteční model. Namísto toho probíhá počáteční inicializace kombinací stavebních bloků. Ty se dělí na matematické operátory, analytické funkce a konstanty. Konkrétní podmnožina těchto primitiv je dána nastavením algoritmu. Jejich kombinací vznikají matematické výrazy, které představují kandidátní řešení a zároveň slouží pro výpočet fitness. Typicky se jako fitness používá středních odchylka vstupních datových bodů od očekávaných hodnot daných navrženým výrazem a cílem je ji minimalizovat. V tabulce 2.2 je ukázka možných vstupních dat. Na obrázku 2.8 lze pro tato data vidět jeden z možných výrazů navržených pomocí GP pou-

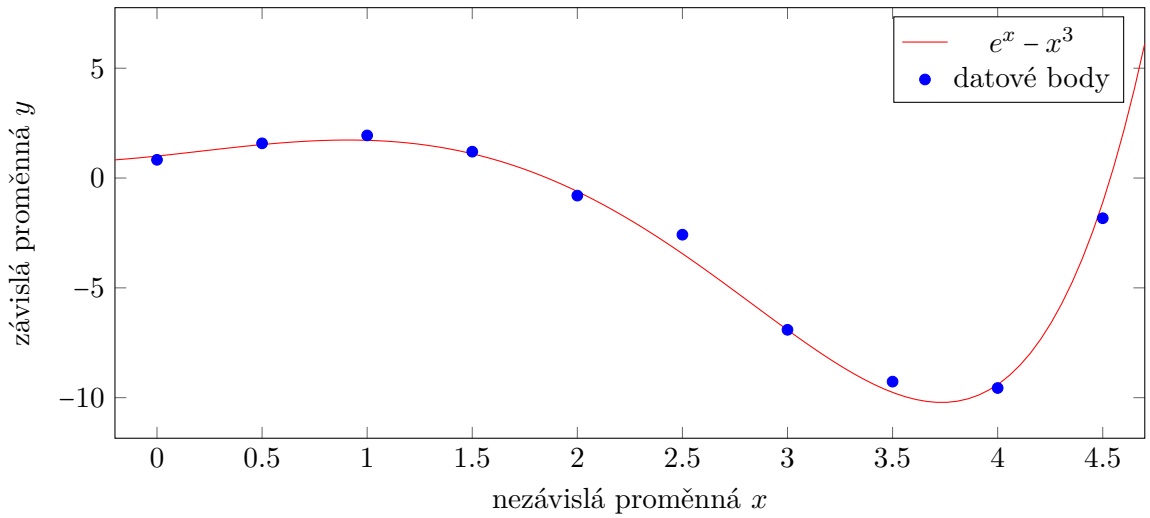
živajícího stromový a lineární chromozom. V grafu na obrázku 2.9 je zobrazeno proložení ukázkových dat zmíněným výrazem.

nezávislá proměnná x	0	0.5	1	1.5	2	2.5	3	3.5	4	4.5
závislá proměnná y	0.83	1.58	1.94	1.2	-0.8	-2.58	-6.91	-9.27	-9.56	-1.83

Tabulka 2.2: Ukázka trénovacích dat pro symbolickou regresi



Obrázek 2.8: Fenotyp $e^x - x^3$ symbolické regrese ve a) stromovém chromozomu a b) mřížce CGP (viz kapitola 3)



Obrázek 2.9: Zobrazení bodů z tabulky 2.2 a jejich proložení funkcí navrženou symbolickou regresí z obrázku 2.8.

Kapitola 3

Kartézské genetické programování

V této kapitole je na základě zdrojů [7], [11], [3] a [5] vysvětleno kartézské genetické programování (CGP), které tvoří základ metody použité v této práci. Přestože zmíněný algoritmus má v názvu „genetické programování“, tak využívá i principů evoluční strategie $(1 + \lambda)$. Používá dvourozměrnou mřížku acyklicky propojených prvků, která je zakódována do lineárního chromozomu. Evoluce hledá takové propojení prvků mřížky, které bude řešit zadanou úlohu. Pro tvorbu potomků používá pouze mutaci a obvykle pracuje s velkým množstvím generací a malou populací. CGP nachází velmi dobré uplatnění při řešení řady problémů, zejména pak při hledání realizace hardwarových prvků. Lze jej využít i při návrhu obrazových filtrů, pro symbolickou regresi, v evolučním umění a dalších úlohách.

3.1 Parametry

Kandidátní řešení CGP jsou definována následujícími parametry:

- n_i: počet primárních vstupů
- n_o: počet primárních výstupů
- n_r: počet řádků mřížky
- n_c: počet sloupců mřížky
- l: levels-back parametr, definuje konektivitu grafu
- n_n: počet vstupů uzlu
- n_f: počet funkcí v množině funkcí
- Γ: konečná množina funkcí

Celkový počet primárních vstupů je závislý na konkrétní řešené úloze. Například při řešení symbolické regrese by se jednalo o dimenzi hledané funkce, při návrhu obrazového filtru by byl počet vstupů dán velikostí jádra filtru a při návrhu sčítačky či násobičky by byl závislý na délce sčítaných/násobených čísel v bitech. Stejným způsobem je na úloze závislý počet primárních výstupů.

Počet řádků a sloupců mřížky dohromady udává celkový počet elementů, které lze ve struktuře zapojit. Jedná se o parametry nezávislé na úloze, které se zadávají před spuštěním algoritmu. Rozměry mřížky mohou ovlivnit, jak složité bude najít řešení a jaké vlastnosti řešení bude mít. V závislosti na požadavcích na hledané řešení se může jednat o výhodné i nežádoucí vlastnosti. Optimální rozměry mřížky je nutné hledat experimentálně.

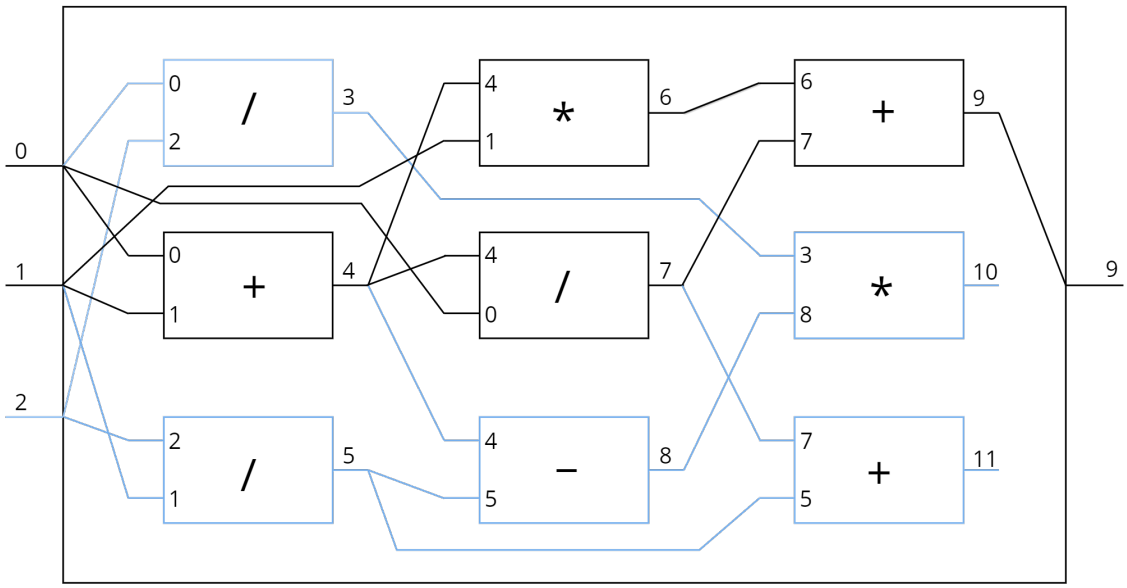
S počtem sloupců mřížky úzce souvisí l-back parametr. Jedná se o počet bezprostředně předcházejících sloupců, ze kterých lze k prvkům v daném sloupci připojit vstupy. Jeho

maximální hodnota je omezena počtem sloupců a značí maximální propojitelnost. Společně s velikostí mřížky může do značné míry ovlivnit, jak rychle bude evoluce konvergovat k řešení a jak bude výsledná navržená struktura vypadat.

Každý prvek realizuje některou z funkcí definovaných v množině funkcí. Vstupem funkce jsou vstupy příslušného uzlu. Při vyšším počtu vstupů do uzlu je možné, aby jednotlivé elementy realizovaly složitější funkce, což zvyšuje míru abstrakce. Množina použitelných funkcí je závislá na úloze, ale do jisté míry ji lze ovlivnit. Ne vždy je výhodné používat plnou množinu všech možných funkcí. Naopak se může vyplatit použít pouze jejich podmnožinu, pokud je známo, že pro návrh řešení je dostačující nebo je z technologických důvodů žádoucí nalézt řešení používající pouze některé funkce.

3.2 Reprezentace

Chromozom kandidátního řešení CGP je řetězec $n_r n_c(n_n+1)+n_o$ celých čísel. Řetězec sestává z k-tic, kde $k = n_r n_c(n_n+1)$, kódujících prvky mřížky a jedné n_o-tice kódující primární výstupy. Každý prvek je kódován jako $(i_1, ..., i_{n_n}, f)$, kde $i_1, ..., i_{n_n}$ jsou indexy vstupů připojených do prvku a f je realizovaná funkce. Primárním vstupům jsou přiřazeny indexy $\langle 0, n_i - 1\rangle$. Výstupy prvků jsou indexovány sekvenčně sloupec po sloupci hodnotami $\langle n_i, n_i + n_r n_c - 1\rangle$ a podle přiřazených indexů jsou prvky seřazeny v chromozomu. Na konci chromozomu je n_o-tice primárních výstupů sestávající z n_o hodnot značící, na který prvek nebo primární vstup je daný primární výstup připojen.



Obrázek 3.1: Kandidátní řešení CGP s parametry $n_r = 3, n_c = 3, l = 3, n_i = 3, n_o = 1, n_n = 2, n_f = 4, \Gamma = \{+, -, *, /\}$ kódované chromozomem (0, 2, 3)(0, 1, 0)(2, 1, 3)(4, 1, 2)(4, 0, 3)(4, 5, 1)(6, 7, 0)(3, 8, 2)(7, 5, 0)(9).

Protože v mřížce CGP musí být zachována acykličnost, tak existuje omezení na připojení vstupů do prvků. K prvkům daného sloupce mohou být připojeny vždy pouze primární vstupy nebo vstupy z l předcházejících sloupců. Nelze propojovat prvky ve stejném sloupci nebo připojit vstupy z následujících sloupců. Na obrázku 3.1 je ukázka zakódovaného řešení. Lze si všimnout několika vlastností tohoto kódování. Některé prvky mohou být použity

vícekrát, zatímco výstupy jiných nemusí být nikam připojeny. Stejně tak nemusí být použity ani některé primární vstupy a prvky mřížky, což vede k jevu specifickému pro CGP, který bude popsán v kapitole 3.4.

3.3 Dekódování chromozomu

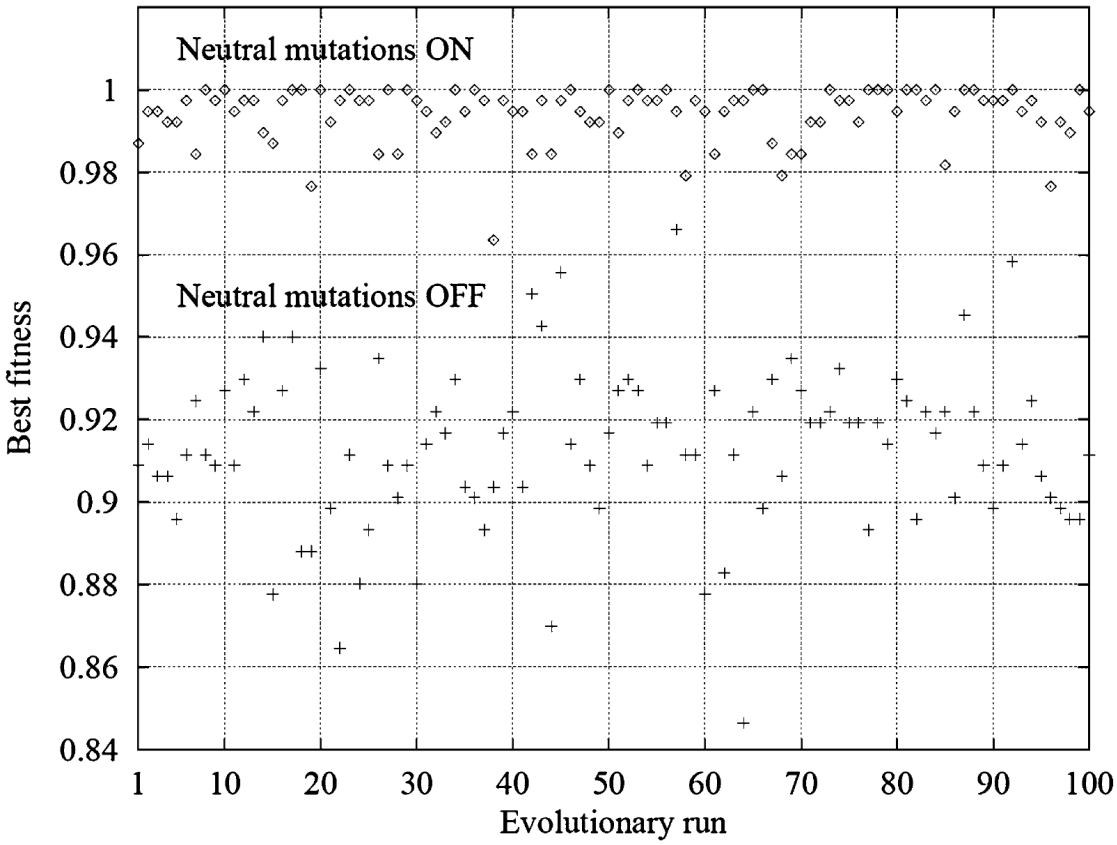
Kandidátní řešení je pro běh algoritmu nutné nejen umět zakódovat, ale také dekódovat pro následné ohodnocení. Protože během celé evoluce dochází k velkému počtu evaluací, které z většiny určují výslednou dobu běhu, je třeba zbytečně nevyhodnocovat nezapojené prvky. Z tohoto důvodu se při dekódování chromozomu CGP postupuje nikoli od vstupů, ale od výstupů. Tímto způsobem je totiž možné pomocí připojených vstupů k prvkům určit, které bloky jsou zapojeny a které nikoli.

V první fázi dekódování se poznačí aktivní bloky. Nejprve se algoritmus podívá, kam jsou zapojeny primární výstupy. Bloky s danými indexy jsou označeny jako aktivní. V dalším kroku se zpracují tyto aktivní bloky a jako aktivní se označí prvky, které jsou připojeny na jejich vstup. Tímto způsobem se pokračuje, dokud se nezpracují všechny prvky. V druhé fázi dochází k vyhodnocování výstupu kandidátního řešení. Tentokrát se postupuje směrem od primárních vstupů. Jsou brány aktivní prvky od nejnižších indexů a vyhodnocovány jejich funkce z připojených vstupů. Výsledek vyvedený na jejich výstup je zaznamenán pod jejich indexem, aby mohl být použit při vyhodnocování dalších prvků. Nakonec je tímto způsobem získán výstup celého programu a je možné vypočítat fitness. V závislosti na úloze může být potřeba druhou fázi opakovat pro všechny případy fitness a fitness hodnotu počítat ze všech získaných výstupů.

3.4 Mutace v CGP

Pro tvorbu nových potomků CGP využívá pouze mutaci, protože křížení se zatím neukázalo jako přínosné [7]. Četnost mutace je stejně jako u ostatních EA nastavitelná pomocí parametru, a je tedy možné mutovat jeden, nebo více genů. Při generování nové hodnoty mutovaného genu je však třeba brát v úvahu omezení plynoucí ze způsobu reprezentace popsané v kapitole 3.2.

Mutace v genotypu CGP se nemusí projevit ani změnou fenotypu či kvality jedince. Jelikož ne všechny bloky mřížky bývají zapojené, může dojít k mutaci právě nezapojeného bloku. Tento typ mutace je nazýván neutrální mutace. Neutrální mutace může rovněž nastat, pokud dojde k mutaci vstupu aktivního prvku, který však není funkcí prvku využíván, nebo se do mutovaného genu vygeneruje stejná hodnota, kterou již měl. Vzniklý jedince má stejnou fitness a fenotyp jako jeho rodič, ale jeho další mutací postupně dochází ke generování nový struktur v nepoužívané části mřížky. Když jsou nakonec tyto nové sturktury zapojeny, dochází k velkým změnám ve fenotypu a potenciálně významnému zvýšení kvality jedince. CGP využívá k selekci elitismus, tedy jako rodič je vybírán nejlepší jedinec z populace. Aby bylo možné využít neutrální mutace, je třeba selekci mírně upravit. Za předpokladu, že nejlepší potomci mají stejnou fitness jako rodič, tak je rodič v příští generaci z populace vyřazen a jako další rodič je vybrán mladší jedinec. Pokud by nebylo toto opatření aplikováno, byly by generované struktury průběžně zahazovány a nemohlo by docházet k velkým změnám mezi generacemi. Značně pozitivní vliv neutrálních mutací byl experimentálně ověřen [7], jak je vidět na obrázku 3.2. Na vodorovné ose tohoto obrázku je pořadí běhu a na svislé ose dosažená fitness nejlepšího jedince v daném běhu. Bylo provede



Obrázek 3.2: Vliv neutrálních mutací na výsledky CGP, převzato z [7]

sto běhů běz využití neutrálních mutací, které jsou označeny křízky. Dále bylo provedeno rovněž sto běhů označených čtverečky, které neutrálních mutací využívají. Řešenou úlohou v tomto experimentu byl návrh korektní tří-bitové paralelní násobičky. Z obrázku je dobře vidět, že bez neutrálních mutací se nedařilo nalézt úplné řešení zadané úlohy a průměrná kvalita řešení byla nižší než při jejich využití. Naopak jejich zapnutí vedlo k vysoké úspěšnosti při návrhu korektního řešení.

3.5 Algoritmus CGP

Samotný algoritmus CGP je jednoduchý. Vychází z evoluční strategie $(1 + \lambda)$, kde většinou platí, že $\lambda = 4$ [7]. Na začátku algoritmu se náhodně vytvoří počáteční populace. Všechny hodnoty v chromozomu musí odpovídat dříve popsaným omezením na propojení prvků. Následně je třeba všechny jedince ohodnotit a, jelikož CGP využívá princip elitismu, vybrat nejlepšího z nich. V dalším kroku se vytváří nová populace. Novou populaci tvoří nejlepší jedinec z minulé generace a jeho λ potomů. Jakmile je nová populace vytvořena, opět se pokračuje ohodnocením jedinců a výběrem rodiče. Algoritmus končí, pokud bylo nalezeno řešení nebo bylo dosaženo maximálního počtu generací. Poloformálně zapsaný algoritmus vypadá následovně:

1. Vygeneruj $1 + \lambda$ náhodných jedinců do počáteční populace
2. Ohodnoť počáteční populaci
3. Dokud není splněna ukončující podmínka:

   3.1. Najdi nejlépe ohodnoceného jedince
   3.2. Vygeneruj λ potomků mutací nalezeného nejlepšího jedince
   3.3. Ohodnoť potomky
   3.4. Vytvoř novou populaci z vybraného rodiče a všech λ potomků
4. Jako řešení vrať jedince s nejlepší fitness

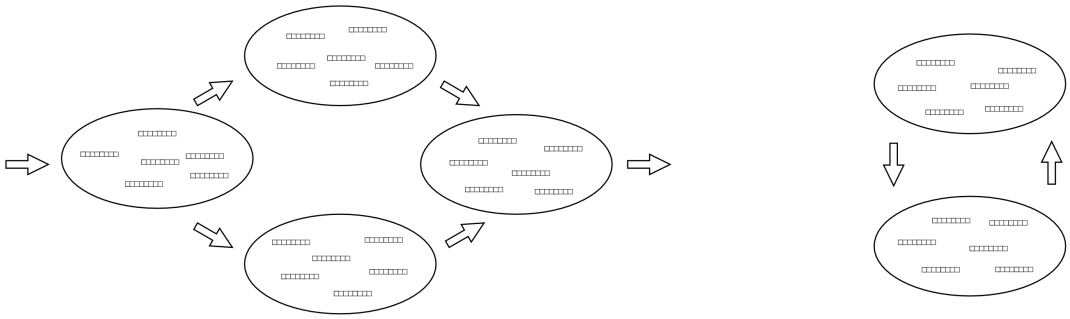
Kapitola 4

Koevoluční algoritmy

Koevoluční algoritmy (CoEA) jsou evoluční algoritmy, ve kterých jsou jedinci porovnáváni na základě výsledků své interakce s ostatními jedinci namísto hodnocení pomocí objektivní fitness funkce. Jedinci mohou interagovat s ostatními jedinci z téže populace nebo s jedinci z jedné, či více jiných populací. Na rozdíl od klasických evolučních algoritmů je ohodnocení subjektivní a je možné, že pořadí podle fitness týchž dvou jedinců se bude během evoluce měnit. Typické úlohy řešení CoEA, podrobnější informace o fitness funkci CoEA a jejich vlastnoti jsou popsány v této kapitole na základě [10], [13] a [4]. Pro kapitolu o aproximaci fitness v CoEA pak byl navíc použit zdroj [14].

4.1 Kooperativní vs. kompetitivní algoritmy

Podle způsobu interakce jedinců můžeme koevoluční algoritmy řadit do dvou tříd: kooperativní a kompetitivní koevoluční algoritmy. V kooperativních algoritmech jsou jedinci pozitivně hodnocení, když dokážou na řešení úlohy dobře spolupracovat s ostatními jedinci a naopak jsou penalizováni, pokud společně podávají slabé výsledky. Naopak u kompetitivních algoritmů jsou jedinci odměňováni na úkor těch, se kterými interagují.



Obrázek 4.1: Koevoluční algoritmus pro a) kompoziční (vlevo) a b) problémy založené na testování (vpravo)

Existuje i novější dělení podle typu problému, který algoritmus řeší. Konkrétně se jedná o kompoziční a problémy založené na testování. Ve své podstatě toto dělení odpovídá uvedenému staršímu dělení. Koevoluce pro kompoziční problémy vychází z kooperativních CoEA. Problém řešený pomocí CoEA je rozdělen na menší části, jejichž řešení jsou vyvíjena v

samostatných populacích a následně skládána do funkčního celku. Při použití koevoluce na problémy založené na testování jsou navržená řešení hodnocena pomocí interakce se sadou testovacích případů. V jedné populaci pak mohou být vyvíjena řešení zadané úlohy, zatímco v druhé populaci se vyvíjí sada testů, která je aplikována na kandidátní řešení pro jejich ohodnocení. Toto dělení je ilustrováno obrázkem 4.1.

4.2 Fitness v koevolučních algoritmech

Hodnocení jedinců v EA lze obecně rozlišovat jako [13]:

- objektivní – hodnocení nezávislé na ostatních jedincích
- subjektivní – hodnocení závislé na ostatních jedincích
- interní – hodnocení ovlivňuje průběh evoluce, což znamená, že jsou na jeho základě vybírání rodiče pro tvorbu potomků. Fitness, jak je běžně chápána, spadá do této kategorie.
- externí – hodnocení průběh evoluce nijak neovlivňuje

Lze tedy říci, že CoEA jsou EA používající subjektivní interní hodnocení pro stanovení fitness. Subjektivní podstata fitness funkce vede k určitým problémům, jak bude popsáno v následující kapitole. Z tohoto důvodu vzniká snaha zavést do CoEA i metodu externího hodnocení. Jedním z navržených řešení je zaznamenávání historie evolučního běhu, ze které je možné diagnostikovat dynamiku systému. Mezi další navržené metody patří informačně-teoretický přístup pro diagnostiku vzniku chování nebo metoda založená na dominanci, která může být užitečná při porovnávání algoritmů.

Protože hodnocení jedinců probíhá na základě jejich vzájemné interakce, vyvstává otázka jak jedince pro tento účel vybírat. Párovat každého s každým je výpočetně příliš náročné, tudíž se nepoužívá. Pro některé problémy, kde je ve fitness funkci málo šumu, je dostačující použít turnajovou metodu s vyřazovacím systémem. U problémů s vyšší mírou šumu lze použít turnajovou metodu s k náhodnými soupeři. Některé vlastnosti úloh však vyžadují jiné sofistikovanější metody.

4.3 Aproximace fitness

Jednou ze základních operací používanou v EA je vyhodnocení fitness. Čas spotřebovaný touto operací v průběhu algoritmu je největší nevýhodou EA. Navíc přesná fitness funkce ani nemusí v některých aplikacích existovat. Z těchto důvodů se používají techniky aproximace fitness funkce. Příkladem časté aplikace aproximace je optimalizace aerodynamiky designu nebo predikce struktury proteinů z důvodu vysokých požadavků na výpočetní čas. Dalším příkladem je evoluční umění, kde chybí explicitní fitness funkce.

První používanou technikou je modelování fitness. Existuje řada modelů, z nichž nepopulárnější jsou polynomiální modely, dopředné neuronové sítě a další metody strojového učení. Je možné použít i metody vzorkování trénovacích dat. Velmi důležitá je přesnost aproximace původní fitness funkce, která je ve většině případů známa a se kterou by měl být model použit dohromady. Je však nutno používat ji efektivně, protože je časově náročná. Dále je potřeba dosáhnou co nejvyšší možné kvality aproximačního modelu. Z tohoto důvodu je nutné brát v úvahu výběr vhodného modelu, použití aktivního vzorkování dat a váhování, výběr trénovací metody a způsob měření odchylky.

Velmi podobným konceptem je predikce fitness, která vyhodnocení fitness nahrazuje zjednodušenou aproximací vyvíjející se společně s řešením. Namísto snahy aproximovat celou fitness funkci cílí na její část danou aktuálním stavem evoluce. Pro vývoj prediktorů společně s řešením se používají CoEA. Prediktory obsahují podmnožinu vstupů, pomocí které je vypočítána fitness hodnota kandidátních řešení, a vyvíjí se společné s populací. Cílem evoluce prediktoru je v podstatě výběr nejdůležitějších vstupů, které mají skutečný dopad na fitness. Není pak nutné se zabývat ostatními vstupy, na kterých by byl zbytečně spotřebován výpočetní čas.

4.4 Výhody a nevýhody koevolučních algoritmů

CoEA mají řadu vlastností, které jim v porovnání s tradičními EA zajišťují výhody nebo naopak je staví do nevýhody. Podstata těchto algoritmů umožňuje řešit určité úlohy, na kterých klasické EA selhávají. První takovou kategorií jsou úlohy s velkým, zejména nekonečným, prohledávacím prostorem. CoEA by mohly být schopny se v něm zaměřit na prohledávání menších relevantních oblastí [13]. Jako příklad si lze uvést návrh řadicí sítě. Při použití klasického EA je nutné vzít statickou podmnožinu všech možných vstupů a na ní populaci trénovat. Tento přístup zajistí, že navržená řadí síť bude dávat velmi dobré výsledky pro použité vstupy. Může se však stát, že při použití této sítě na jiné vstupy budou výsledky velmi neuspokojivé. Pokud je však na návrh řadicí sítě použit CoEA, je umožněno spolu se sítí nechat algoritmus navrhnout i trénovací vstupy. Ty se v průběhu evoluce mění a zdokonalují, takže i výsledná síť by měla mít větší schopnost řadit i ty případy, které nebyly vefitness funkci testovány.

Druhou skupinou úloh jsou problémy, ve kterých neexistuje přirozené objektivní hodnocení. Často se jedná např. o herní strategie, mezi kterými existují netranzitivní vztahy, což znamená, že existují stratregie A, B, C takové, že A je preferováno před B, B je preferováno před C a C je preferováno před A. Zejména pokud se takové vztahy vyskytují v celém prohledávaném prostoru strategií, pak není jasné, jaké vlastnosti nejlepší strategie má. Je to taková strategie, která porazí co největší množství libovolných strategií? Nebo by měla naopak porazit raději menší množství dobrých strategií? V klasických EA je tento problém neřešitelný, protože vyžadují objektivní míru hodnocení. Naopak CoEA jsou vhodné, protože ze své podstaty používají míru subjektivní.

Posledním typem úloh, kde CoEA mají oproti klasickým EA výhodu, jsou komplexní vysoce strukturované problémy. U EA je nutné celou strukturu vyvíjet najednou, což se může ukázat jako značně obtížné. Při použití CoEA je však možné úlohu rozdělit na komponenty, které jsou následně vyvíjeny oddělené v různých populacích. Jejich řešení je tudíž mnohem efektivnější. CoEA však mají několik dalších výhod. Pokud se jedinci v jedné populaci mírně změní, následuje adaptace v dalších populacích. K těmto kompetitivním adaptacím dochází opakovaně, až je nalezeno velmi kvalitní řešení, které je potenciálně lepší než u jiných EA. Zajímavou vlastností je také to, že CoEA přirozeně směřují k nalezení Nashova ekvilibria [13]. Nalezená řešení jsou tedy co nejméně dominovaná jinými.

CoEA však trpí i řadou problémů. Reálné aplikace kompetitivních i kooperativních algoritmů často selhávají, protože dynamika koevolučních systémů je často velmi složitá a neintuitivní. Tato vlastnost ve spojení se subjektivitou hodnocení dělá měření pokroku obtížné. Není možné rozhodnout, jestli specifikované míry kvality jedince značí vývoj k lepšímu. Ze stejného důvodu je složitá i diagnóza chování systému. Vyskytuje se vyšší citlivost na určité vlastnosti úloh, která souvisí s vyskytujícím se negativním chováním CoEA. Je známo několik typů nepříznivého chování algoritmu. Jedním je ztráta gradientu, při které

dojde k situaci, kdy jedna populace značně dominuje druhé. Druhá populace pak nemá dostatek informací k učení a adaptaci. Další chování, které může nastat, je cyklické chování. Jedna populace získá nad druhou výhodu a druhá se následně přizpůsobí. Tento proces se několikrát opakuje, až nakonec první populace dospěje do původní podoby. V takovém případě nedojde k žádnému zlepšení. Podobným chováním je průměrná stabilita. Při výskytu tohoto jevu se populace cyklicky nebo na stálo stabilizuje ve značně suboptimálním bodě prostoru. Toto chování je pro koevoluční systémy přirozené, ale nežádoucí. Může dojít i k problémům s přílišnou specializací řešení na slabiny soupeře. Takové řešení je velmi křehké, protože nedokáže fungovat obecně, a po odstranění dané slabiny začne podávat slabé výsledky.

Kapitola 5

Koevoluce při návrhu klasifikátoru obrazu

Klasifikace je úloha rozpoznávání vzorů z oblasti strojového učení, při které jsou objekty tříděny do předem definovaných tříd. Třídění se provádí na základě předchozího učení na trénovacím vzorku dat, u kterých je známa jejich příslušnost ke třídě. Pro natrénování systému se tedy používá učení s učitelem. Algoritmus implementující klasifikaci se nazývá klasifikátor. Jedním z druhů objektů určený pro klasifikaci může být obraz. Klasifikace obrazu je jednou ze základních úloh počítačového vidění. V současti se pro její řešení nejčastěji používají hluboké konvoluční neuronové sítě [9]. Toto řešení však vyžaduje dostatek výpočetních zdrojů, což znamená jeho nezanedbatelnou spotřebu elektrické energie. Uplatnění v nízkopříkonových aplikacích je z tohoto důvodu obtížné. Cílem této práce je pomocí genetického programování navrhnout jednoduchý klasifikátor, který by měl pro tento typ aplikací dobrý poměr spotřeby a spolehlivosti. V minulosti proběhla snaha o návrh takového klasifikátoru obrazu pomocí CGP [2] s dobrými výsledky. Metoda použitá v této práci spojuje CGP s aproximací fitness pomocí koevolučního algoritmu, jak bylo navrženo v [4] a [3].

5.1 Evoluční návrh klasifikátoru

Návrh klasifikátoru pro klasifikační úlohu o c třídách lze řešit dvěma způsoby. Lze navrhnout jeden klasifikátor s c výstupy, který dokáže klasifikovaný objekt zařadit do příslušné třídy. Druhou možností je návrh c binárních klasifikátorů, které rozhodují, zda objekt do jejich třídy patři, či nikoli. Spojením takových binárních klasifikátorů lze dostat klasifikaci do jedné ze tříd. Výstupy klasifikátorů mohou nabývat binárních, celočíselných nebo reálných hodnot. Každý výstup pak dává pravděpodobnost, se kterou objekt do dané třídy patří. Obraz, se kterým se v této práci pracuje, je šedotónový, a proto bude používán 8-bitový datový typ. Z tohoto důvodu mohou výstupy navrhovaného klasifikátoru nabývat hodnot od 0 do 255. Minimální hodnota značí, že obraz do třídy jistě nepatří a naopak maximální hodnota znamená, že obraz do třídy jistě patří. Konečnou třídu určenou klasifikátorem udává výstup s nejvyšší hodnotu. Pokud by výstupů se stejnou nejvyšší hodnotou bylo více, obraz se považuje za neklasifikovaný. Pro velikost vstupního obrazu $m \times n$ pixelů má klasifikátor $m \times n$ primárních vstupů. Do klasifikátoru tedy vstupuje celý obraz.

Pro účely evoluce vypadá fitness funkce klasifikátoru (CGP programu) následovně. Index vstupního vzorku označíme jako j a výstup klasifikátoru po zpracování tohoto vzorku jako C_j. Jestliže obraz patří do třídy reprezentované příslušným výstupem, pak se na něm očekává hodnota $E_j = 255$, jinak $E_j = 0$. Fitness hodnota se vypočítá jako střední odchylka mezi očekávaným a obdrženým výstupem pro každý z k trénovacích vstupů. Vztah pro binární klasifikátor vypadá takto:

$$f = \frac{1}{k} \sum_{j=1}^{k} |E_j - C_j|. \tag{5.1}$$

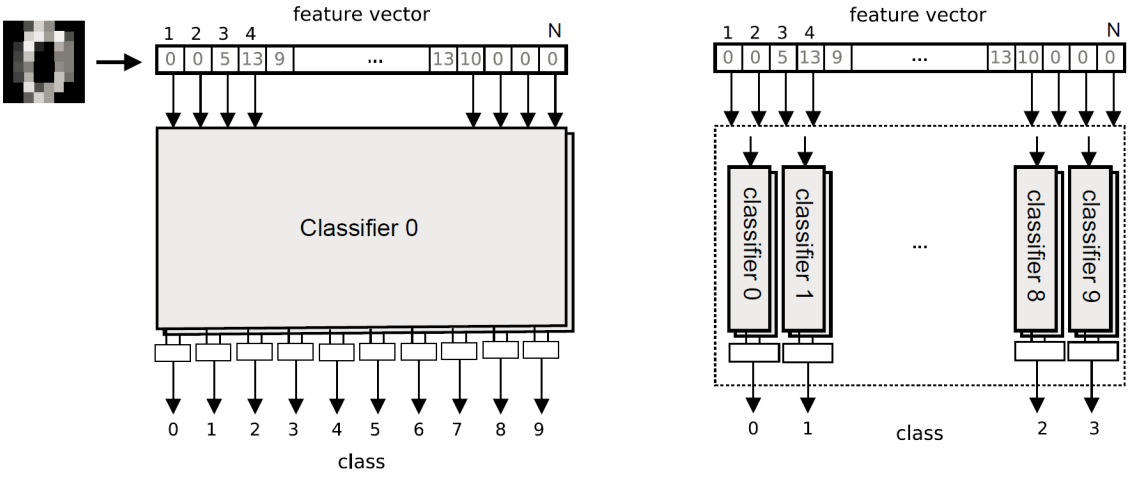
Pro klasifikátor rozpoznávající více tříd absolutní hodnota odpovídá součtu odchylek na všech jeho výstupech. Upravený vztah pro indexy výstupů $\langle 1, c \rangle$ pak vypadá následovně.

$$f = \frac{1}{k} \sum_{j=1}^{k} \sum_{n=1}^{c} |E_{j_n} - C_{j_n}| \tag{5.2}$$

Do množiny funkcí pro jednotlivé elementy budou použity 8-bitové funkce z tabulky 5.1. Jejich vhodnost pro práci s obrazem byla již v minulosti ověřena [7], [2].

Index	**Funkce**	**Popis**	**Index**	**Funkce**	**Popis**
0	255	konstanta	6	$min(x, y)$	minimum
1	x	identita	7	$x + y$	součet
2	$255 - x$	inverze	8	$x +^s y$	součet se saturací
3	$x \gg 1$	dělení dvěma	9	$(x + y) \gg 1$	průměr
4	$x \gg 2$	dělení čtyřmi	10	$(x > 127)?x : y$	podmíněné přiřazení
5	$max(x, y)$	maximum	11	$\lvert x - y \rvert$	absolutní rozdíl

Tabulka 5.1: Funkce použité pro návrh klasifikátoru

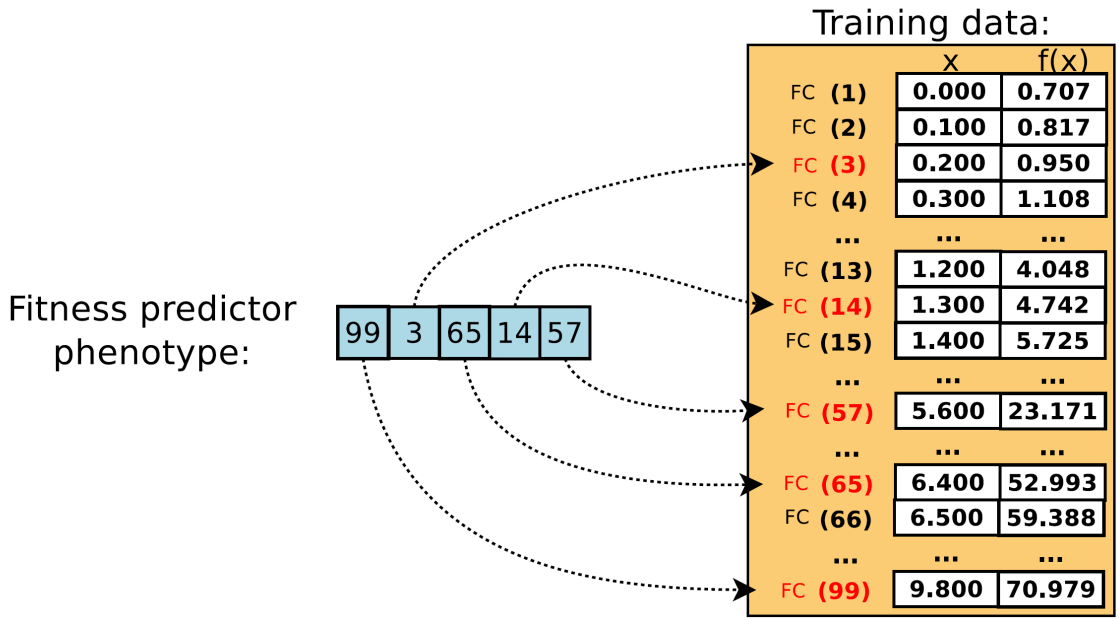


Obrázek 5.1: Klasifikace číslic pomocí a) klasifikátoru s deseti výstupy (vlevo) b) deseti binárních klasifikátorů (vpravo), převzato z [2]

5.2 Koevoluční návrh klasifikátoru obrazu

Z důvodů snížení časové náročnosti je v této práci přístup pomocí klasického CGP, jak byl popsán v předchozí kapitole, kombinován s predikcí fitness pomocí koevolučního algoritmu. Predikce fitness umožňuje použít podmnožinu trénovacích dat, která se nazývá prediktor, k ohodnocení kandidátního řešení. Délka prediktorů může být pevná, případně se může adaptovat v průběhu evoluce. V této práci je použita první možnost. Využití predikce fitness pomocí prediktorů umožňuje několikanásobně snížit čas potřebný pro evoluci řešení, zatímco kvalita řešení zůstává srovnatelná [3]. Při evoluci klasifikátoru obrazu by měla být úspora času významná, protože pro jeho ohodnocení pomocí fitness funkce je potřeba velké množství trénovacích dat.

Jak již bylo zmíněno, množinu trénovacích dat T úlohy lze nahradit prediktorem P. Množina T obsahuje k fitness případů, tedy $k = |T|$. Prediktor P je malou podmnožinou množiny T, $P \subset T$ a platí, že $|P| \ll |T|$. Prediktor obsahuje několik fitness případů, resp. odkazů na ně, které se použijí k predikci fitness. Cílem prediktoru je poskytnout takovou predikci, která je rychlá a dokáže rozlišovat mezi libovolnými páry kandidátních programů. Aby bylo možné prediktory vyvíjet, musíme být schopni vypočítat jejich fitness. K tomuto účelu se používají trénovací řešení. Ohodnocení se vypočítá jako absolutní střední odchylka mezi objektivní a subjektivní fitness těchto trénovacích řešení. Archiv trénovacích řešení obsahuje vybraná kandidátní řešení, která se objevila v populaci v průběhu evoluce.



Obrázek 5.2: Prediktor, převzato z [4]

Pro výpočet fitness fitness hodnot během evoluce s prediktory je nutné mít dvě verze fitness funkce. První je objektivní fitness $f_{obj}(s, T)$, která používá všechna data z trénovací množiny, a druhá je subjektivní $f_{subj}(s, P)$, které používá pouze data z prediktoru P. Tedy:

$$f_{obj}(s, T) = \frac{1}{k} \sum_{j=1}^{k} g(y_j(s, T)), \tag{5.3}$$

$$f_{subj}(s, P) = \frac{1}{m} \sum_{j=1}^{m} g(y_j(s, P)), \tag{5.4}$$

kde $y_j(s, T)$ je odezva kandidátního programu s na j-tý vstup z trénovací množiny T a m je počet trénovacích případů v prediktoru. Funkce g je závislá na řešené úloze a pro klasifikaci obrazu výplývá z fitness funkcí definovaných vztahy 5.1, 5.2.

$$g(C_j) = |E_j - C_j| \tag{5.5}$$

pro binární klasifikátor, resp.

$$g(C_j) = \sum_{n=1}^{c} |E_{j_n} - C_{j_n}| \tag{5.6}$$

pro klasifikátor s klasifikací do c tříd.

Kromě ohodnocení kandidátních programů musíme být schopni vypočítat i fitness samotného prediktoru, aby bylo možné jej vyvíjet vedle řešení úlohy. Jestliže archiv trénovacích programů označíme A, pak je fitness prediktoru P vyjádřena následovně:

$$f_p(P, A) = \frac{1}{u} \sum_{i=1}^{u} |f_{obj}(A_i, T) - f_{subj}(A_i, P)| \tag{5.7}$$

kde A_i je i-tý trénovací vstup a u je počet programů v archivu trénovacích programů.

5.3 Algoritmus

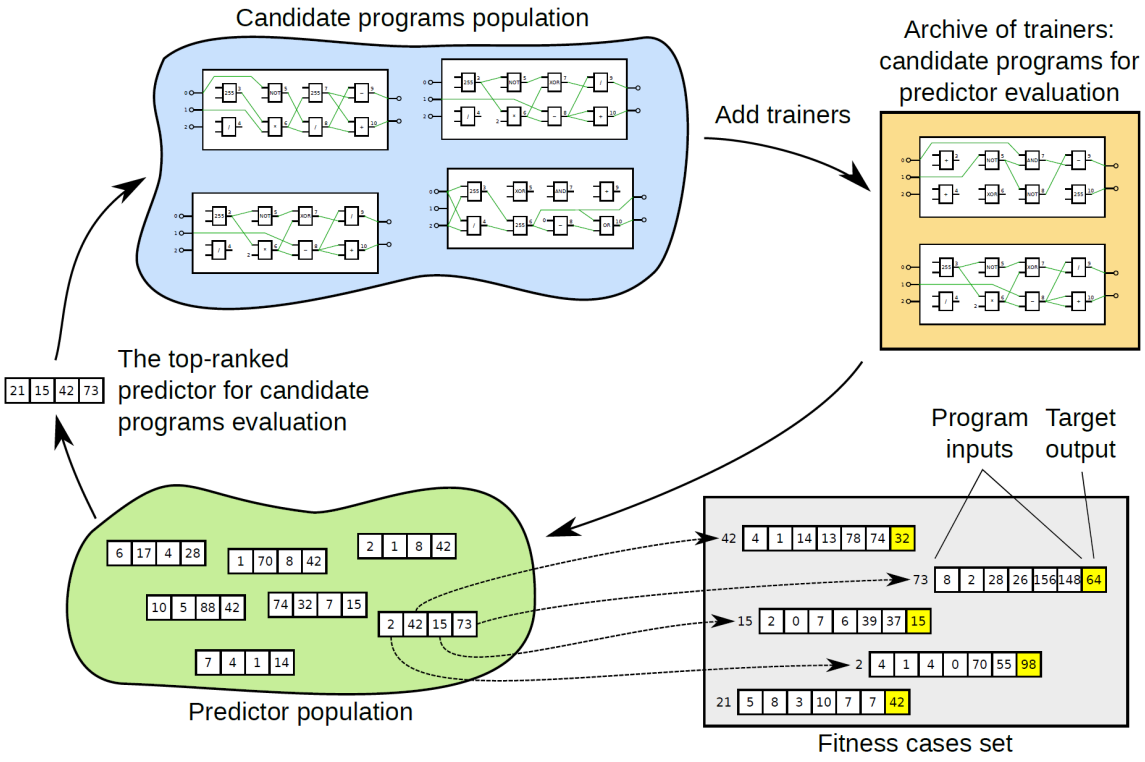
Koevoluční algoritmus (algoritmus 1) pro návrh klasifikátorů sestává z inicializace algoritmu a dvou evolučních procedur. V jedné jsou vyvíjena kandidátní řešení a v druhé prediktory. Každá z těchto procedur běží ve vlastním vlákně. Jako první je spuštěna metoda inicializace, ve které jsou nejprve inicializovány populace programů a prediktorů vygenerováním náhodných jedinců. Počáteční programy jsou následně použity pro inicializaci archivu. Pokud má být archiv větší než je velikost populace programů, je doplněn o další náhodně vygenerované jedince. Po dobu běhu algoritmu jsou obsahem archivu nejlepší nalezení jedinci, ale zároveň je jeho část vyhrazena i pro náhodně jedince, kteří zajišťují diverzitu. Po inicializaci je archiv ohodnocen pomocí objektivní fitness funkce (vztah 5.3) a populace prediktorů pomocí fitness funkce pro prediktory (vztah 5.7). Nejlepší aktuální program a prediktor jsou zaznamenány pro další použití. Nejlepší program je rovněž označen za řešení úlohy a příznak ukončení algoritmu je nastaven na false.

Po dokončení inicializace je spuštěna samotná evoluce programů paralelně s evolucí prediktorů. Metoda evoluce programů je prováděna následovně. Na jejím začátku je ohodnocena populace programů subjektivní fitness funkcí (vztah 5.4) pomocí nejlepšího známého prediktoru. Pokud v současné generaci došlo ke zlepšení, je daný program vložen do archivu trénovacích programů. Je vypočítána jeho objektivní fitness, která se porovná s objektivní fitness programu označeného za řešení. Pokud i v tomto případě došlo ke zlepšení, je řešení aktualizováno nejlepším programem ze současné generace. Následně je vytvořená nová populace a celý postup se opakuje pro zvolený počet generací. Nakonec je nastaven příznak ukončení na true, aby se ukončila všechna vlákna algoritmu, a jako výstup je vráceno řešení. Evoluce programů je založena na CGP, a proto přebírá všechny jeho vlastnosti.

Algoritmus 1 Pseudokód pro koevoluci populace programů pomocí CGP a populace prediktorů fitness

```
1: procedure Init
2:     progPop.init()
3:     predPop.init()
4:     archive.init()
5:
6:     for all t in archive do
7:         t.fitness = evalObj(t)
8:     end for
9:     for all p in predPop do
10:        p.fitness = evalPred(p)
11:    end for
12:
13:    topPred = findTopRanked(predPop)
14:    topProg = findTopRanked(archive)
15:    solution = topProg
16:    finished = false
17: end procedure
18:
19: procedure ProgramEvolution
20:    for gen = 1 to genMax do
21:        for all s in progPop do
22:            s.fitness = evalSubj(s)
23:        end for
24:        prevTopProg = topProg
25:        topProg = findTopRanked(progPop)
26:
27:        if topProg.fitness() > prevTopProg.fitness() then
28:            archive.insert(topProg)
29:            if evalObj(topProg) > solution.fitness() then
30:                solution = topProg
31:            end if
32:        end if
33:
34:        progPop.newPop()
35:    end for
36:    finished = true
37: end procedure
```

```
38: procedure PredictorEvolution
39:     while not finished do
40:         for all p in predPop do
41:             p.fitness = evalPred(p)
42:         end for
43:         topPred = findTopRanked(predPop)
44:         predPop.newPop()
45:     end while
46: end procedure
47:
48: procedure Coevolution
49:     Init()
50:     ProgramEvolution and PredictorEvolution
51: end procedure
```



Obrázek 5.3: Koevoluční algoritmus s prediktory, převzato z [4]

Pro účel zdokonalování predikce fitness je využívána metoda evoluce prediktorů. V podstatě se jedná o jednoduchý evoluční algoritmus. Populace prediktorů je ohodnocena na archivu trénovacích programů. Z populace je vybrán prediktor s nejlepší fitness a je označen za nejlepší prediktor, který se dále bude používat pro ohodnocování programů. Následně se vytvoří nová populace pomocí jednobodového křížení a mutace, která má pravděpodobnost 20%. Tyto kroky se opakují, dokud není nastaven příznak ukončení. Obě populace se synchronizují nepřímo, protože jedna generace prediktorů připadá na několik generací

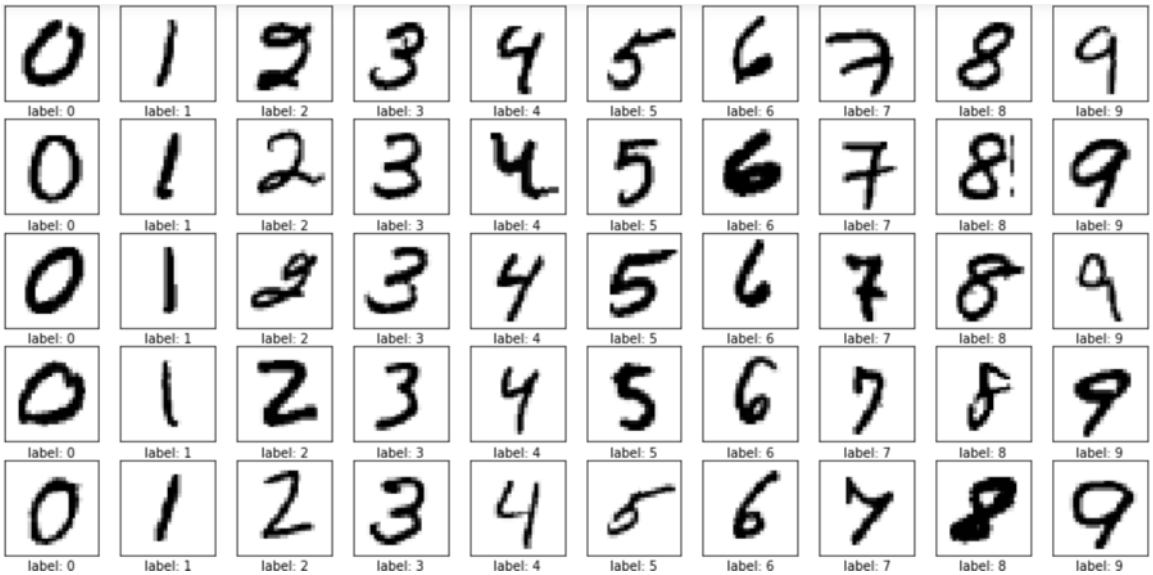
programů. Přesný počet generací závisí na velikosti archivu a počtu prediktorů. V obou případech platí, že čim vyšší hodnota, tím více generací programů se stihne vyhodnotit na jednu generaci prediktorů. Grafické znázornění celého koevolučního algoritmu je na obrázku 5.3.

Kapitola 6

Nastavení experimentů

Cílem experimentů je navrhnout nejlepší možný klasifikátor ručně psaných číslic pomocí koevolučního algoritmu, který byl popsán v kapitole 5. Byly popsány dva možné přístupy k evolčnímu návrhu klasifikátoru. Pro tyto experimenty byl vybrán přístup rozdělení klasifikátoru na menší úlohy, resp. na binární klasifikátory pro jednotlivé třídy. Výsledný klasifikátor je nakonec složen z řešení pro podúlohy jednotlivých číslic. Tento přístup byl zvolen, protože poskytuje kvalitnější výsledky. Přesnost i úplnost takto navrženého klasifikátoru je značně vyšší než u klasifikátoru, kde bylo řešení navrhováno jako jeden celek [2].

Datovou sadou použitou v této práci je databáze ručně psaných číslic MNIST [6], která je široce požívaná pro trénování a testování v oblasti strojového učení. Sestává z normalizovaných šedotónových obrazů o rozměrech 28×28 pixelů spadající do jedné z deseti tříd $0 - 9$. Databáze obsahuje celkem 70 000 obrazů, které jsou rozděleny do trénovací sady o 60 000 obrazech a testovací sady čítající zbývajících 10 000 obrazů. Vzorek databáze je vidět na obrázku 6.1. Pro účely experimentů byla celá datová sada podvzorkována na velikost 14×14 pixelů.



Obrázek 6.1: Vzorek MNIST databáze [1]

Aby bylo možné docílit kvalitních výsledků koevoluce, je nutné správně zvolit její parametry. Parametry pro CGP byly zvoleny na základě již provedených experimentů s klasifikací číslic [2]. Jejich nastavení tedy vypadá následovně. Mřížka CGP má jediný řádek o 100 prvcích. L-back parametr byl nastaven na maximální hodnotu. Toto uspořádání umožňuje sestavit řešení více způsoby, protože omezení dané pravidly na propojení prvků mezi jednotlivými sloupci je minimální. Nalezení řešení je díky tomu evoluci usnadněno. Každý prvek může být nastaven na jednu funkci z tabulky 5.1. Četnost mutace je nastavena na hodnotu 3, operátor mutace může tedy pro tvorbu nového potomka změny až 3 geny z původního rodičovského chromozomu. Pro populaci byla zvolena velikost 5 jedinců.

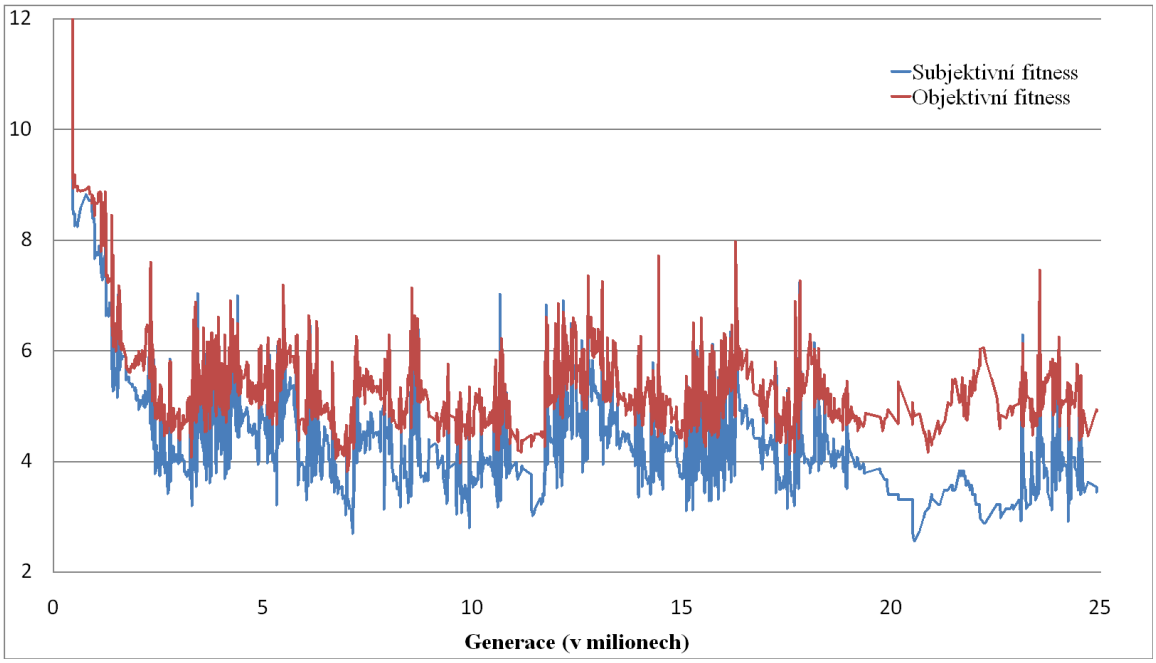
Hodnoty pro zbylé parametry koevoluce byly hledány pomocí prvotních experimentů s různým nastavením. Vybrány poté byly ty hodnoty parametrů, které vedly k nejlepší fitness na konci běhu. Z důvodu vysoké výpočetní náročnosti a omezeného času na diplomovou práci však nebylo možné spustit dostatečný počet běhů pro nalezení plně optimálních parametrů. Nejlepší nalezené parametry jsou následující. Velikost populace prediktorů byla nastavena na 30 a velikost archivu na 105 jedinců. Tyto dvě hodnoty znamenají, že nový prediktor je získán přibližně každých 600 generací CGP. Prediktor obsahuje 3000 testovacích případů, což je 5% trénovací datové sady. Oproti klasickému CGP je tedy koevoluce s tímto nastavením 20x rychlejší. Běh je ukončen po dosažení 25 000 000 generací CGP.

Pro každou třídu bylo spuštěno 8 běhů algoritmu. Výsledný klasifikátor byl tudíž následně sestavován na základě 80 běhů. Experimenty byly provedeny na čtyřech školních serverech edesign1 - edesign4.

Kapitola 7

Výsledky experimentů

V této kapitole jsou prezentovány výsledky získané koevolucí s nastavením popsaným v předešlé kapitole. Nejprve je zhodnocen průběh koevoluce a jednotlivé binární klasifikátory z hlediska kvality řešení a použitých prvků na základě evolučních běhů provedených nad trénovací sadou obrazů. Poté jsou vyhodnoceny výsledky konečného složeného klasifikátoru pro všechny třídy číslic. Tyto výsledky byly získány pomocí testovací sady MNIST o 10 000 obrazů. Následuje porovnání s jinými přístupy ke klasifikaci obrazu a možnosti vylepšení přístupu pomocí koevoluce, které by mělo vést k návrhu klasifikátorů s vyšší přesností klasifikace než která byla dosažena v této práci.



Obrázek 7.1: Typický vývoj subjektivní fitness nejlepšího jedince v populaci a jí odpovídající objektivní fitness v průběhu koevoluce. Rozsah osy y je omezena pro lepší čitelnost.

Na obrázku 7.1 je graf s průběhem typického běhu koevoluce. Znázorněn je zde vývoj subjektivní fitness nejlepšího jedince a jí odpovídající objektivní fitness. Je vidět, že na začátku běhu fitness rapidně klesá. Následuje fáze, kdy dochází k postupnému zlepšování řešení. Po většinu generací však fitness osciluje okolo určité hodnoty fitness. Jednou

za několik milionů generací se takto podaří nalézt lepší řešení. Lze si také všimnout, že se snižující se fitness nalazeného řešení dochází ke zvětšování chyby predikce. Subjektivní fitness má tendenci být nižší než objektivní fitness. To značí částečnou specializaci kandidátního řešení na používaný prediktor. Za účelem zlepšování fitness by bylo vhodné v této fázi zvětšit velikost prediktoru, aby docházelo k přesnější predikci. Tato možnost bude podrobněji popsána v kapitole 8. Při pohledu na minima průběhu objektivní fitness si je možné také všimnout, že posledních cca 15 milionů generací nemusí být s použitým nastavením vůbec produktivních. Stejné výsledky by tak mohly být dosaženy snížením počtu generací např. na 10 milionů a ušetřený čas by pak bylo možné použít na více běhů.

	Fitness$_{obj}$		**Fitness$_{subj}$**		**Počet operací**		**Počet vstupů**	
Třída	Průměr	Nejlepší	Průměr	Nejlepší	Průměr	Nejlepší	Průměr	Nejlepší
0	3,7603	3,5704	3,4512	5,0103	28	33	21,75	23
1	2,6363	2,4856	2,2825	2,4963	35,5	38	29	31
2	6,3265	5,9328	5,6906	4.8507	32,625	35	27	30
3	8,5516	7,7954	7,799	7,14	30,75	32	25,25	25
4	6,9526	6,5481	6,5313	6,0027	33,375	28	27	23
5	6,3876	6,1266	5,8466	5,185	36,875	46	27,125	33
6	4,957	4,6356	4,6562	4.1137	31,625	35	26,625	31
7	6,0179	5,4023	5,3622	4.435	38,75	46	31,125	39
8	10,1299	9,333	9,4153	7.905	28,25	31	24,5	27
9	9,2556	8,7973	8,7808	8.4667	34,25	36	29,5	31

Tabulka 7.1: Objektivní a subjektivní fitness, počet operací a počet vstupů nejlepších navržených klasifikátorů a průměrné hodnoty ze všech experimentálních běhů provedených nad trénovací sadou obrazů.

V tabulce 7.1 je vidět porovnání nejlepších nalezených binárních klasifikátorů s průměrem ze všech běhů. Z dosažených hodnot fitness, resp. středních odchylek, vyplývá, že klasifikátory pro různé číslice je různě obtížně navrhnout. Nejjednodušší je řešení úlohy pro číslici 1 a jen o něco málo těžší pro číslici 0. Naopak nejsložitější je návrh klasifikátoru pro třídu 8, která je následovaná třídou 9. Tyto dva klasifikátory vykazují téměř čtyřnásobně vyšší střední odchylku oproti řešením pro 0 a 1. Při porovnání objektivních a subjektivních fitness hodnot jednotlivých řešení se potvrzuje pozorování z grafu na obrázku 7.1. Subjektivní fitness je ve většině případů nižší než objektivní. Klasifikátory jsou tedy skutečně lépe přizpůsobeny obrazům z prediktoru než zbytku datové sady. Z pohledu počtu prováděných operací jsou navržená řešení poměrně kompaktní. Kombinovaná velikost je 360 prvků. Lze si také všimnout, že nejlepší nalezená řešení provádí naprůměrné množství operací nad vstupním obrazem. Výjimkou je pouze klasifikátor pro číslici 4. Stejný trend je pozorovatelný u počtu vstupů do klasifikátoru, což ovšem částečně souvisí s počtem operací.

Na obrázku 7.2 jsou vidět prováděné operace nejlepším klasifikátorem pro třídu 0. Klasifikátory ostatních tříd jsou k nahlédnutí v příloze A. Pozorovat lze, že mezi nejčastěji používané funkce patří minimum, součet se saturací, průměr a absolutní rozdíl. Velmi málo jsou naopak zastoupeny jednovstupové funkce. Pokud již jsou použity, pak je to nejčastěji inverze nebo dělení čtyřmi.

Pro lepší názornost jsou použité primární vstupy jednotlivých binárních klasifikátorů zobrazeny na obrázku 7.3. Je zde vidět, že klasifikátory sledují rysy specifické pro jednotlivé

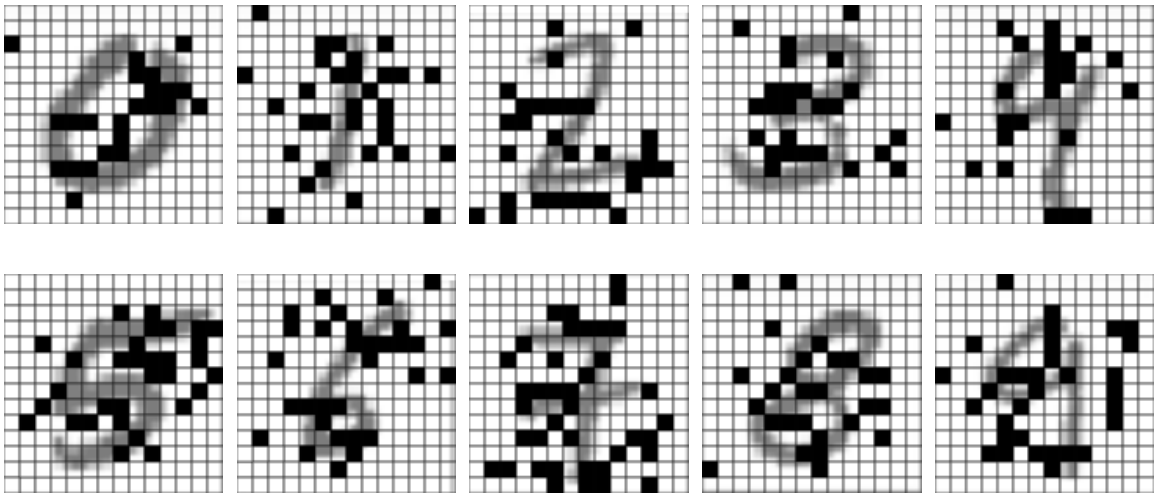
číslice. Například pro číslici 4 je zřejmě sledováno spodní zakončení nožičky a mezera mezi tahy ve vrchní části obrazu. U některých číslic je vidět, že tvar tvořený použitými vstupy sleduje tvar číslice. Jedná se o číslice 7 nebo 5 a některé další. Naopak u číslice 1 se zdá, že klasifikátor se spíše snaží vyloučit možné tahy v jejím okolí. Částečně je tato tendence vidět i u jiných číslic.

---

def $class0(p)$:

$e_0 = F_{11}(p_{92}, p_{132})$; $e_1 = F_5(p_{94}, p_{65})$; $e_2 = F_9(p_{39}, p_{79})$; $e_3 = F_6(p_{133}, p_{91})$;
$e_4 = F_{10}(e_2, p_{79})$; $e_5 = F_6(p_{103}, p_{102})$; $e_6 = F_6(p_{144}, p_{145})$; $e_7 = F_2(p_{28})$;
$e_8 = F_6(e_6, p_{143})$; $e_9 = F_2(p_{119})$; $e_{10} = F_8(e_9, e_5)$; $e_{11} = F_{11}(e_4, p_{80})$; $e_{12} = F_6(e_{10}, p_{101})$;
$e_{13} = F_6(e_1, p_{81})$; $e_{14} = F_2(e_{13})$; $e_{15} = F_6(e_0, p_{64})$; $e_{16} = F_8(e_{12}, e_{15})$; $e_{17} = F_3(e_3)$;
$e_{18} = F_5(e_8, e_{16})$; $e_{19} = F_1(e_{11})$; $e_{20} = F_8(e_{17}, e_{19})$; $e_{21} = F_2(p_{96})$; $e_{22} = F_6(p_{119}, p_{105})$;
$e_{23} = F_8(p_{50}, p_{172})$; $e_{24} = F_8(e_{20}, e_{14})$; $e_{25} = F_{10}(e_{24}, e_{21})$; $e_{26} = F_{10}(e_{18}, e_7)$;
$e_{27} = F_{11}(e_{23}, e_{23})$; $e_{28} = F_6(e_{27}, e_{13})$; $e_{29} = F_{10}(e_{22}, e_{28})$; $e_{30} = F_6(e_{29}, e_{25})$;
$e_{31} = F_6(e_{26}, e_{30})$; $e_{32} = F_{11}(e_{30}, e_{31})$;
return e_{32}

Obrázek 7.2: Nejlepší klasifikátor pro třídu 0



Obrázek 7.3: Použité primární vstupy v nejlepších nalezených binárních klasifikátorech. Klasifikátorem použité vstupy jsou vyznačeny černě. Číslice na pozadí jsou náhodně vybrané obrazy z databáze MNIST.

Následující data byla získána již po kombinaci jednotlivých binárních klasifikátorů do jednoho klasifikátoru pro všechny třídy. Pro jejich vyhodnocení je třeba nejprve definovat několik pojmů a vztahů.

- **True positive (TP)** – počet objektů náležících do třídy, které do ní byly i klasifikovány.
- **True Negative (TN)** – počet objektů nenáležících do třídy, které do ní byly klasifikovány.

- **False Positive (FP)** – počet objektů nenáležících do třídy, které do ní však byly klasifikovány.
- **False Negative (FN)** – počet objektů náležících do třídy, které do ní však nebyly klasifikovány.
- **Přesnost** – podíl relevantních objektů mezi všemi klasifikovanými. Je dána vztahem

$$\text{Přesnost} = \frac{TP}{TP + FP}$$

- **Úplnost** – podíl relevantních objektů, které byly klasifikovány. Je dána vztahem

$$\text{Úplnost} = \frac{TP}{TP + FN}$$

Sledován byl nejen počet správně zařazených obrazů, ale také počet nesprávně klasifikovaných a neklasifikovaných. Podrobně je tato statistika vidět v tabulce 7.2. Na první pohled je zřejmé, že většina obrazů byla klasifikována správně. Značná část však byla označena jako neklasifikovaná. Počet nesprávně zařazených obrazů je malý. Při bližším prozkoumání si lze všimnout, že nejvyšší úspěšnosti dosahl klasifikátor při rozpoznávání číslice 1 a následně 0. Naopak nejhorších výsledků klasifikátor dosahl u číslice 8. Při porovnání s fitness binárních klasifikátorů toto není překvapující a výsledky jsou odpovídající. Z počtu nesprávně klasifikovaných obrazů vyčnívá hodnota 48, která udává počet obrazů s číslicí 9 rozpoznaných jako 4. Množství chyb v tomto konkrétním případě je několikanásobně vyšší než u všech ostatních případů. Jestliže vezmeme v potaz, nakolik podobný tvar mají tyto dvě číslice v psané podobě, není zmíněná chybovost překvapivá. Klasifikátor musí umět rozeznávat niance, aby byl schopen je správně rozeznat s velmi vysokou spolehlivostí blízkou člověku. Zdaleka nejnižší chybovost má číslice 1. Nejen, že jsou obrazy patřící do této třídy velmi zřídka zařazeny do jiné třídy, ale zároveň není časté, že by klasifikátor rozeznal jako 1 některou z ostatních číslic. Procento neklasifikovaných obrazů této třídy je taktéž nejnižší. Z právě rozebraných údajů lze vypočítat přesnost a úplnost klasifikátoru podle uvedených vztahů. Hodnoty na hlavní diagonále tabulky jsou TP. Ostatní hodnoty na některém z řádků klasifikařních tříd patří do FP. Neklasifikované obrazy patří do FN. Vypočítan přesnost a úplnost jsou uvedeny v tabulce 7.3. Přesnost je jak pro jednotlivé číslice, tak pro celou testovací sadu vysoká. Ve všech případech přesahuje 90%. Nejvyšší je opět pro číslici 1, ale nejhorší pro 9. Zde se projevují těžkosti rozeznání 4 a 9. Celková přesnost klasifikátoru je 95,9%. U úplnosti jsou hodnoty nižší a rozmanitější. Zatímco u některých číslic, přesněji opět u 1 a 0, přesahují 90%, v nejhorším případě úplnost nedosahuje ani 70%. Problémem je opět číslice 8. Celková úplnost je 80,22%. Nižší hodnoty tohoto ukazatele jsou způsobeny především množstvím neklasifikovaných obrazů.

Při celkovém pohledu na všechna data je zřejmé, že kvalita výsledného klasifikátoru je z velké části přímo závislá na fitness jednotlivých binárních klasifikátorů. Jelikož jsou však tyto klasifikátory vyvíjeny zcela odděleně, nedokáže ani výborná fitness s jistotou zajistit rozlišení podobných číslic. Z tohoto důvodu je hlavním zdrojem chybovosti označení obrazu jako neklasifikovaného. K tomuto může dojít ve dvou případech. Buďto klasifikátor rozhodne, že číslice určitě nepatří ani do jedné třídy, nebo, že patří do více tříd.

	Číslice									
Třída	0	1	2	3	4	5	6	7	8	9
0	912	0	3	1	0	6	6	2	4	2
1	0	1075	2	0	1	0	3	4	4	5
2	1	2	824	19	1	1	2	12	5	0
3	2	1	5	729	1	5	0	5	8	0
4	2	0	1	0	746	0	12	3	3	6
5	3	1	0	4	1	661	5	0	7	2
6	9	0	2	2	8	12	788	0	8	1
7	2	1	10	7	2	2	0	820	2	6
8	0	2	3	10	3	1	2	4	666	4
9	1	0	2	2	48	8	1	10	5	801
n/a	48	53	180	236	171	196	139	168	262	182

Tabulka 7.2: Zařazení číslic do klasifikačních tříd výsledným klasifikátorem. Sloupce představují skutečnou třídu zařazovaného obrazu a řádky třídu, které byla určena klasifikátorem.

Číslice	**Přesnost**	**Úplnost**
0	0,9744	0,9306
1	0,9826	0,9471
2	0,9504	0,7984
3	0,9643	0,7218
4	0,9651	0,7597
5	0,9664	0,741
6	0,9494	0,8225
7	0,9624	0,7977
8	0,9583	0,6838
9	0,9123	0,7939
Celkem	0,959	0,8022

Tabulka 7.3: Přesnost a úplnost výsledného klasifikátoru pro všechny třídy

7.1 Srovnání přístupů

Důležitou součástí zhodnocení dosažených výsledků je porovnání s výsledky jiných přístupů k návrhu řešení. Experimentálně získané výsledky této práce budou srovnány nejprve s přístupem používajícím klasické CGP [2]. Poté bude následovat srovnání s klasifikátorem realizovaným pomocí neuronové sítě [8]. V obou případech byla pro návrh použita databáze číslic MNIST, takže výsledky nejsou ovlivněny výběrem datové sady.

V této práci byly pro CGP část koevoluce použity stejné parametry jako v případě práce využívající klasické CGP [2]. Rozdíl byl v dostupných hardwarových prostředcích. Z tohoto důvodu je značný rozdíl v počtu provedených experimentů. Zatímco v této práci bylo provedeno dohromady 80 spuštění algoritmu, výsledky práce zabývající se CGP jsou postaveny na 1000 bězích. Při vyšším počtu spuštění má algoritmus více šancí najít kvalitnější řešení. Toto mohlo do určité míry ovlivnit kvalitu výsledného nejlepšího řešení. Hlavními ukazateli kvality klasifikátoru jsou již zmíněná přesnost a úplnost. Koevolucí bylo dosaženo přesnosti

95,9% a úplnosti 80,22%. Při použití CGP byly tyto hodnoty 94,6%, resp. 86,1% [2]. Použitím koevoluce tedy bylo dosaženo o něco vyšší přesnosti, ale úplnost je nižší. V obou případech byla hlavním zdrojem chyb neschopnost klasifikátoru zařadit některé obrazy. V případě koevoluce je však podíl neklasifikovaných obrazů značně vyšší, konkrétní poměr mezi těmito dvěma metodami je 1635:904. Pokud je ale již obraz zařazen do jedné z tříd, je vyšší pravděpodobnost, že je klasifikován správně. Vypočítaná přesnost i úplnost klasifikátorů získaných z obou metod jsou srovnatelné. Byl tedy potvrzen předpoklad založený na výsledcích jiných úloh [4], že pomocí koevoloce lze běh algoritmu mnohonásobně urychlit a zároveň zachovat kvalitu výsledků.

Druhým srovnávaným přístupem je klasifikace pomocí plně propojené vícevrstvé (konkrétně třívrstvé) perceptronové síťe [8], která má 28×28 vstupů, 300 skrytých neuronů a 10 výstupů. Její trénování na databázi MNIST trvá řádově desítky minut. S touto sítí bylo dosaženo 97,67% přesnosti klasifikace na 8-bitové reprezentaci. Z uvedeného počtu neuronů lze vypočítat počet operací, které musí síť provést pro výpočet jednoho výstupu. Jelikož se jedná o plně propojenu síť, tak je počet operací $28 \times 28 \times 300 + 300 \times 10 = 238200$. Prováděnými operacemi je hlavně násobení kvůli váhování jednotlivých vstupů, ale samotné neurony provádí i další operace. V porovnání s klasifikátorem navrženým v této práci, který provádí 360 jednoduchých operací (součet, bitový posuv či přiřazení), je vypočítané množství násobení u neuronové sítě mnohonásobně vyšší. Násobení je navíc náročnější operace. Díky komplexitě a velikosti neuronových sítí není možné je bez velkých obtíží použít v nízkopříkonových hardwarových aplikacích. Při dnešním trendu minimalizace zařízení a jejich spotřeby tento problém narůstá. Je tedy zřejmé, že evolučně navržený klasifikátor má značně nižší spotřebu elektrické energie a je menší. Z tohoto důvodu je jašně vidět jeho uplatnění. Výhodou neuronové sítě však zůstává značně kratší doba trénování a vyšší spolehlivost.

Kapitola 8

Možnosti vylepšení

I přes dobré dosažené výsledky má metoda použitá v této práci a nastavení provedených experimentů určité nedostatky. Jejich odstraněním by mělo být možné navrhovat ještě spolehlivější klasifikátory. Při provádění experimentů pro získání optimálního nastavení experimentů se ukázalo, že velikost prediktoru je nejdůležitějším parametrem koevoluce. Jeho nesprávné nastavení vede v lepším případě ke ztrátě urychlení běhu algoritmu, které koevoluce oproti CGP poskytuje, jestliže je jeho hodnota příliš vysoká. V opačném lepším případě je prediktor příliš malý na to, aby bylo možné nalézt uspokojivé řešení úlohy. Zejména pokud evoluce trvá dlouho, je hledání nastavení značně náročné na čas. Negativní vliv hodnoty parametru se totiž může projevit až po velkém množství generací. Právě velikost prediktoru má tendenci se projevit zastavením zlepšování řešení až po dosažení určité hladiny fitness. Pro řešení tohoto problému lze použít proměnnou délku prediktoru. Koevoluce je v takovém případě modifikována tak, aby se zároveň s obsahem prediktoru vyvíjela i jeho velikost. Evoluce si uchovává záznam o délce prediktoru a modifikuje ho na základě několika pravidel. Změnšování či zvětšování prediktoru se rozhoduje podle rychlosti evoluce. Sleduje se zlepšení fitness za určitý počet generací. Jestliže se fitness řešení zlepšuje, pak se velikost prediktoru navýší, aby byla predikce fitness přesnější. Pokud je detekována stagnace, pak se velikost prediktoru změnší. Tento případ často nastává, když se evoluce dostane do lokálního optima. Zmenšení prediktoru pomáhá lokální optimum opustit. Poslední možností je, že dochází ke snižování fitness. V takovém případě je velikost prediktoru opět snížena. K této situaci často dochází po zmenšení velikosti prediktoru v důledku stagnace. Znamená to, že evoluce opouští lokální optimum. Další zmenšení prediktoru tento proces urychluje. Tento proces pomáhá najít optimální délku prediktoru a zároveň zajišťuje postup evoluce. Bylo ukázáno [3] [4], že v porovnání s koevolucí využívající konstantní velikost prediktoru je touto metodou dosahováno srovnatelných výsledků. Velkou výhodou adaptivního prediktoru je však zejména ušetření času potřebného pro nalezení optimální délky prediktoru s pevnou délkou. Použití prediktorů s proměnnou délkou zajišťuje konvergenci k optimální délce bez ohledu na počáteční nastavenou velikost. Ještě markantnější jsou výhody ve srovnání s klasickým CGP. Koevoluce s adaptivními prediktory vede k častějšímu nalazení řešení v kratším reálném čase a menším počtu generací. Z těchto důvodů by zavedení této modifikace pro návrh klasifikátoru obrazu bylo značným přínosem. Bylo by tak možno dosáhnout spolehlivější klasifikace.

Návrh binárních klasifikátorů není dokonalý, zejména pro některé číslice je nalezení řešení problematické. Zároveň jsou si některé číslice podobné, což způsobuje problémy při spolupráci těchto klasifikátorů v jednom celku. Projevem je zařazení obrazů jako neklasifikovaných, protože více vícstupů má stejnou hodnotu. Pokud by tedy bylo v těchto případech

možné klasifikátor naučit, aby byl schopen rozhodnout, vedlo by to k lepším výsledkům. Zde se proto nabízí možnost rozdělit evoluci na dvě části. První část by byla evoluce binárních klasifikátorů, jak byla provedena v této práci. Po provedení požadovaného počtu běhů a sestavení klasifikátoru pro všechny třídy by se spustila druhá evoluce. Ta by měla za úkol zdokonalit výsledný klasifikátor. Opět je možné provést sadu běhů a vybrat nejlepší řešení.

Jedním ze sledovaných parametrů je i velikost navržených řešení, protože je jím dána spotřeba po nasazení. Čím méně operací musí klasifikátor vykonat, tím nižší je jeho spotřeba a cena. Je tedy možné řešení analyzovat a navrhnout úpravy pro snížení počtu operací pomocí konvenčních metod, nebo je možné opět využít evoluci. Při použití evoluce se sleduje použitých prvků. Všichni jedinci v jejím průběhu, kterým oproti původnímu řešení klesne fitness, jsou vyřazeni z populace. Pracuje se pouze s řešeními, která mají fitness stejnou nebo lepší. Jako rodič se používá vždy jedinec s nejnižším počtem používaných operací. Tento proces vede k úspoře na velikosti řešení.

Možnosti evolučního návrhu jsou samozřejmě také do určité míry omezeny dostupnými prostředky. Jejich množství udává, kolik běhů algoritmu o kolika generacích je možné spustit za jednotku času. Jelikož bylo zjištěno, že zatímco návrh klasifikátorů pro některé číslice je v podstatě bezproblémový a pro jiné náročný, bylo by pravděpodobně přínosnější věnovat více prostředků na návrh zmíněných složitějších řešení. Když se podaří zlepšit fitnes například pro číslici 8, lze očekávat větší zlepšení výsledného řešení, než kdyby se podařilo snížit fitness klasifikátoru třídy 1, který má již i takto malou chybovost.

Při kombinaci popsaných návrhů na vylepšení použité metody by bylo možné ušetřit čas nebo ho využít na další potenciální zlepšení řešení. Výsledné řešení by navíc bylo kvalitnější.

Kapitola 9

Závěr

Cílem tohoto projektu bylo navrhnout metodu využívající genetického programování pro návrh klasifikátoru obrazu, následně tuto metodu implementovat a provést s ní sadu experimentů, které povedou k úspěšnému návrhu řešení. K tomtuto účelu byla nastudována a popsána problematika evolučních algoritmů, zejména pak genetického programování. Konkrétně použitým algoritmem je kartézské genetické programování. Z důvodu potřebného výpočetního času je tento algoritmus kombinován s aproximací fitness pomocí koevolučního algoritmu, což jsou další nastudované a popsané oblasti. Výsledná koevoluční metoda využívá fitness prediktory. Tato metoda byla implementována a experimenty byly provedeny na sadě ručně psaných číslic. Evoluce byla spuštěna pro jednotlivé třídy číslic s cílem navrhnout binární klasifikátory, které byly následně sestaveny do výsledného klasifikátoru všech tříd. Z důvodu vysoké časové náročnosti jednoho běhu a omezeného času na vypracování diplomové práce nebylo možné optimálně nastavit všechny parametry koevoluce, což pravděpodobně do určité míry negativně ovlivnilo výsledky. Přesto se však povedlo najít klasifikátor, který je kvalitou srovnatelný s klasifikátorem získaným pomocí samotného CGP. Výsledný klasifikátor číslic navržený v této práci dosahuje přesnosti 95,9% a úplnosti 80,22%. Využití koevoluce tedy oproti CGP přineslo redukci spotřebovaného času, konkrétně byla použitá metoda 20x rychlejší, a zároveň zachovalo kvalitu řešení. Ušetřený čas je možné využít např. pro vyšší počet generací v každém běhu, což umožní prohledání větší části prohledávaného prostoru řešení a potenciálně nalezení lepšího řešení. Bylo popsáno několik možných modifikací metody jako adaptivní prediktory nebo využití evoluce pro zkvalitnění výsledného řešení, které by měly vést ke spolehlivější klasifikaci. Jejich aplikací je možné pokračovat v této práci.

Literatura

[1] *MNIST database.*
URL http://blog.welcomege.com/mnist-database/

[2] Bidlo, M.; Vašíček, Z.: *On Evolution of Multi-Category Pattern Classifiers Suitable for Embedded Systems.* In *Genetic Programming, 17th European Conference, EuroGP 2014*, LNCS 8599, Berlin, DE: Springer Verlag, 2014, ISBN 978-3-662-44302-6, s. 234–245.
URL http://www.fit.vutbr.cz/research/view_pub.php.cs?id=10482

[3] Drahošová, M.: *Coevolution of Fitness Predictors in Cartesian Genetic Programming.* Dizertační práce, Ústav počítačových systémů, Fakulta informačních technologií, Vysoké učení technické, Brno, CZ, 2017.

[4] Drahošová, M.; Sekanina, L.; Wiglasz, M.: *Adaptive Fitness Predictors in Coevolutionary Cartesian Genetic Programming.* Technická zpráva, Fakulta informačních technologií, Vysoké učení technické, Brno, CZ.

[5] Hajná, K.: *Kartézské genetické programování s dynamickou modifikací parametrů.* Bakalářská práce, Fakulta informačních technologií, Vysoké učení technické, Brno, CZ, 2015.

[6] LeCun, Y.; Cortes, C.; Burges, C. J.: *The MNIST database of handwritten digits.*
URL http://yann.lecun.com/exdb/mnist

[7] Miller, J. F.: *Cartesian Genetic Programming.* Natural computing series, New York: Springer, 2011, ISBN 978-3-642-17310-3.

[8] Mrázek, V.; Sarwar, S. S.; Sekanina, L.; aj.: Design of Power-Efficient Approximate Multipliers for Approximate Artificial Neural Networks. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, Association for Computing Machinery, 2016, ISBN 978-1-4503-4466-1, s. 811–817.
URL http://www.fit.vutbr.cz/research/view_pub.php.cs?id=11142

[9] Rawat, W.; Wang, Z.: *Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. In* Neural Computation*, ročník 29, Cambridge, Massachusetts, USA: MIT Press, 2017, str. 2352–2449.*

[10] Rozenberg, G.; Bäck, T.; Kok, J. N.: Handbook of natural computing. New York: Springer, 2012, ISBN 978-3-540-92911-6.

[11] Sekanina, L.; Vašíček, Z.; Růžička, R.; aj.: Evoluční hardware: Od automatického generování patentovatelných invencí k sebemodifikujícím se strojům. *Edice Gerstner, Academia, 2009, ISBN 978-80-200-1729-1, 328 s.*

[12] Veřmiřovský, J.: Koevoluce v evolučním návrhu obvodů. Diplomová práce, Fakulta informačních technologií, Vysoké učení technické, Brno, CZ, 2016.

[13] Wiegand, R. P.: An Analysis of Cooperative Coevolutionary Algorithms. Dizertační práce, George Mason University, Fairfax, VA, USA, 2003.

[14] Yin, Y.; Sendhoff, B.: Fitness approximation in evolutionary computation - a survey. In GECCO'02 Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, San Francisco, CA, USA: Morgan Kaufmann Publishers, 2002, ISBN 1-55860-878-8, s. 1105–1112.*

Příloha A

Nejlepší navržené binární klasifikátory

---

def $class0(p)$:

$e_0 = F_{11}(p_{92}, p_{132})$; $e_1 = F_5(p_{94}, p_{65})$; $e_2 = F_9(p_{39}, p_{79})$; $e_3 = F_6(p_{133}, p_{91})$;
$e_4 = F_{10}(e_2, p_{79})$; $e_5 = F_6(p_{103}, p_{102})$; $e_6 = F_6(p_{144}, p_{145})$; $e_7 = F_2(p_{28})$;
$e_8 = F_6(e_6, p_{143})$; $e_9 = F_2(p_{119})$; $e_{10} = F_8(e_9, e_5)$; $e_{11} = F_{11}(e_4, p_{80})$; $e_{12} = F_6(e_{10}, p_{101})$;
$e_{13} = F_6(e_1, p_{81})$; $e_{14} = F_2(e_{13})$; $e_{15} = F_6(e_0, p_{64})$; $e_{16} = F_8(e_{12}, e_{15})$; $e_{17} = F_3(e_3)$;
$e_{18} = F_5(e_8, e_{16})$; $e_{19} = F_1(e_{11})$; $e_{20} = F_8(e_{17}, e_{19})$; $e_{21} = F_2(p_{96})$; $e_{22} = F_6(p_{119}, p_{105})$;
$e_{23} = F_8(p_{50}, p_{172})$; $e_{24} = F_8(e_{20}, e_{14})$; $e_{25} = F_{10}(e_{24}, e_{21})$; $e_{26} = F_{10}(e_{18}, e_7)$;
$e_{27} = F_{11}(e_{23}, e_{23})$; $e_{28} = F_6(e_{27}, e_{13})$; $e_{29} = F_{10}(e_{22}, e_{28})$; $e_{30} = F_6(e_{29}, e_{25})$;
$e_{31} = F_6(e_{26}, e_{30})$; $e_{32} = F_{11}(e_{30}, e_{31})$;
return e_{32}

Obrázek A.1: Nejlepší klasifikátor pro třídu 0

---

def $class1(p)$:

$e_0 = F_9(p_{175}, p_{145})$; $e_1 = F_9(p_{49}, p_{63})$; $e_2 = F_2(p_1)$; $e_3 = F_9(p_{107}, p_{93})$; $e_4 = F_5(e_3, e_1)$;
$e_5 = F_9(p_{103}, p_{134})$; $e_6 = F_9(p_{33}, p_{136})$; $e_7 = F_4(p_{49})$; $e_8 = F_{11}(p_{68}, p_{129})$;
$e_9 = F_9(e_5, p_{86})$; $e_{10} = F_8(p_{47}, p_{78})$; $e_{11} = F_6(p_{34}, p_{62})$; $e_{12} = F_9(p_{139}, e_0)$;
$e_{13} = F_{11}(e_9, e_6)$; $e_{14} = F_6(p_{37}, e_{12})$; $e_{15} = F_2(p_{74})$; $e_{16} = F_9(e_{14}, p_{65})$;
$e_{17} = F_9(p_{158}, p_{66})$; $e_{18} = F_8(e_8, e_{13})$; $e_{19} = F_5(e_{11}, e_{16})$; $e_{20} = F_9(e_{18}, e_{15})$;
$e_{21} = F_5(e_7, p_{105})$; $e_{22} = F_{11}(e_{10}, e_4)$; $e_{23} = F_{11}(e_{19}, e_{22})$; $e_{24} = F_6(e_{23}, p_{121})$;
$e_{25} = F_5(e_1, e_{17})$; $e_{26} = F_6(e_{24}, e_{25})$; $e_{27} = F_8(p_{91}, e_{21})$; $e_{28} = F_{11}(p_{56}, e_{26})$;
$e_{29} = F_8(e_{20}, e_{28})$; $e_{30} = F_4(e_{27})$; $e_{31} = F_4(p_{194})$; $e_{32} = F_8(e_{30}, e_{29})$;
$e_{33} = F_8(e_{31}, e_{32})$; $e_{34} = F_{11}(e_{33}, p_{184})$; $e_{35} = F_{10}(e_{34}, e_2)$; $e_{36} = F_8(e_{35}, e_{35})$;
$e_{37} = F_6(p_{102}, e_{36})$;
return e_{37}

Obrázek A.2: Nejlepší klasifikátor pro třídu 1

def $class2(p)$:

$e0 = F_7(p_{176}, p_{72})$; $e_1 = F_9(p_{117}, p_{132})$; $e_2 = F_9(p_{91}, p_{90})$; $e_3 = F_9(p_{170}, e_2)$;
$e_4 = F_7(p_{19}, p_{128})$; $e_5 = F_5(p_{164}, p_{89})$; $e_6 = F_9(p_{47}, e_4)$; $e_7 = F_6(p_{137}, p_{151})$;
$e_8 = F_9(e_6, p_{123})$; $e_9 = F_9(e_7, p_{150})$; $e_{10} = F_1(p_{101})$; $e_{11} = F_5(e_9, p_{134})$;
$e_{12} = F_9(e_0, p_{172})$; $e_{13} = F_6(e_9, p_{184})$; $e_{14} = F_6(e_8, e_{13})$; $e_{15} = F_6(e_{12}, e_5)$;
$e_{16} = F_9(p_{191}, p_{174})$; $e_{17} = F_6(e_{14}, p_{164})$; $e_{18} = F_6(e_{16}, e_3)$; $e_{19} = F_5(e_1, e_{17})$;
$e_{20} = F_9(p_{89}, e_{18})$; $e_{21} = F_6(e_{19}, e_{11})$; $e_{22} = F_{11}(e_{20}, p_{100})$; $e_{23} = F_9(p_{87}, e_{10})$;
$e_{24} = F_{11}(p_{24}, p_{175})$; $e_{25} = F_{10}(e_{23}, p_{182})$; $e_{26} = F_6(e_{22}, p_{152})$; $e_{27} = F_{11}(e_{15}, p_{88})$;
$e_{28} = F_9(e_{21}, e_{27})$; $e_{29} = F_{11}(e_{23}, e_{25})$; $e_{30} = F_8(e_{28}, e_{26})$; $e_{31} = F_8(e_{30}, e_{24})$;
$e_{32} = F_{10}(e_{31}, e_{29})$; $e_{33} = F_{10}(e_{32}, p_{173})$; $e_{34} = F_{11}(e_{32}, e_{33})$;
return e_{34}

Obrázek A.3: Nejlepší klasifikátor pro třídu 2

def $class3(p)$:

$e_0 = F_9(p_{133}, p_{152})$; $e_1 = F_9(p_{79}, p_{92})$; $e_2 = F_7(p_{149}, p_{144})$; $e_3 = F_{11}(p_{115}, p_{132})$;
$e_4 = F_6(p_{47}, p_{137})$; $e_5 = F_{11}(p_{33}, e_4)$; $e_6 = F_9(p_{50}, e_2)$; $e_7 = F_9(p_{91}, e_6)$; $e_8 = F_6(e_1, p_{124})$;
$e_9 = F_6(e_8, e_4)$; $e_{10} = F_5(p_{117}, p_{130})$; $e_{11} = F_6(p_{89}, p_{88})$; $e_{12} = F_5(p_{74}, e_{11})$;
$e_{13} = F_8(e_5, p_{76})$; $e_{14} = F_9(e_9, e_3)$; $e_{15} = F_6(p_{75}, e_{12})$; $e_{16} = F_6(p_{132}, e_{10})$;
$e_{17} = F_6(e_{16}, e_{13})$; $e_{18} = F_6(p_{131}, e_0)$; $e_{19} = F_8(e_7, e_{14})$; $e_{20} = F_6(e_{15}, e_{18})$;
$e_{21} = F_6(e_{20}, p_{87})$; $e_{22} = F_6(e_{21}, e_{17})$; $e_{23} = F_3(e_{11})$; $e_{24} = F_5(e_{19}, e_{22})$;
$e_{25} = F_{11}(e_{19}, e_{24})$; $e_{26} = F_6(e_{25}, e_{23})$; $e_{27} = F_9(e_{25}, e_{12})$; $e_{28} = F_{11}(p_9, p_{16})$;
$e_{29} = F_8(e_{26}, e_{27})$; $e_{30} = F_{10}(e_{29}, e_{28})$; $e_{31} = F_8(e_{30}, e_{30})$;
return e_{31}

Obrázek A.4: Nejlepší klasifikátor pro třídu 3

def $class4(p)$:

$e_0 = F_6(p_{50}, p_{74})$; $e_1 = F_9(e_0, p_{189})$; $e_2 = F_9(p_{116}, p_{120})$; $e_3 = F_{11}(p_{82}, e_1)$;
$e_4 = F_{11}(p_{35}, p_{190})$; $e_5 = F_4(p_{102})$; $e_6 = F_2(p_{77})$; $e_7 = F_3(e_2)$; $e_8 = F_9(p_{53}, p_{103})$;
$e_9 = F_{11}(p_{144}, p_{33})$; $e_{10} = F_6(p_{63}, p_{142})$; $e_{11} = F_9(e_{10}, p_{49})$; $e_{12} = F_{11}(p_{37}, p_{64})$;
$e_{13} = F_3(e_6)$; $e_{14} = F_7(e_5, e_7)$; $e_{15} = F_5(e_{12}, e_3)$; $e_{16} = F_9(e_{13}, e_{14})$;
$e_{17} = F_4(e_8); e_{18} = F_{11}(p_{18}, e_{11})$; $e_{19} = F_6(e_{15}, e_{18})$; $e_{20} = F_8(e_4, e_{19})$;
$e_{21} = F_8(e_{16}, e_{20})$; $e_{22} = F_{11}(p_{191}, p_{20})$; $e_{23} = F_8(e_{21}, e_{17})$; $e_{24} = F_8(e_{23}, e_9)$;
$e_{25} = F_{10}(e_{24}, e_{22})$; $e_{26} = F_8(e_{25}, e_{25})$; $e_{27} = F_{11}(p_{98}, e_{26})$;
return e_{27}

Obrázek A.5: Nejlepší klasifikátor pro třídu 4

def $class5(p)$:

$e_0 = F_9(p_{101}, p_{74})$; $e_1 = F_6(p_{37}, p_{51})$; $e_2 = F_9(p_{94}, p_{110})$; $e_3 = F_9(p_{128}, p_{93})$;
$e_4 = F_9(p_{133}, p_{58})$; $e_5 = F_6(p_{77}, e_4)$; $e_6 = F_7(e_1, e_0)$; $e_7 = F_7(p_{64}, p_{130})$;
$e_8 = F_5(p_{117}, p_{161})$; $e_9 = F_5(e_8, p_{131})$; $e_{10} = F_9(e_6, p_{161})$; $e_{11} = F_5(p_{119}, p_{94})$;
$e_{12} = F_6(p_{148}, p_{35})$; $e_{13} = F_3(p_{52})$; $e_{14} = F_9(e_{11}, e_9)$; $e_{15} = F_9(p_{82}, p_{118})$;
$e_{16} = F_6(e_{10}, e_{13})$; $e_{17} = F_5(e_{12}, e_7)$; $e_{18} = F_9(e_5, e_{17})$; $e_{19} = F_6(e_3, p_{123})$;
$e_{20} = F_9(e_{18}, p_{78})$; $e_{21} = F_6(p_{80}, p_{79})$; $e_{22} = F_9(p_{163}, p_{101})$; $e_{23} = F_{11}(e_{16}, e_{20})$;
$e_{24} = F_9(e_{14}, e_2)$; $e_{25} = F_2(e_{21})$; $e_{26} = F_8(e_{25}, p_{67})$; $e_{27} = F_5(e_{15}, e_{19})$;
$e_{28} = F_9(p_{55}, p_{68})$; $e_{29} = F_9(e_{22}, e_{25})$; $e_{30} = F_9(e_{28}, p_{54})$; $e_{31} = F_9(p_{95}, e_{24})$;
$e_{32} = F_9(e_{30}, e_{26})$; $e_{33} = F_1(e_{29})$; $e_{34} = F_{11}(e_{32}, p_{82})$; $e_{35} = F_{11}(e_{33}, e_{14})$;
$e_{36} = F_5(p_{97}, e_{21})$; $e_{37} = F_6(e_{27}, e_{31})$; $e_{38} = F_2(e_{36})$; $e_{39} = F_6(e_{35}, e_{23})$;
$e_{40} = F_{10}(e_{39}, e_{34})$; $e_{41} = F_4(e_{40})$; $e_{42} = F_9(e_{37}, e_{41})$; $e_{43} = F_{10}(e_{42}, e_{38})$;
$e_{44} = F_8(e_{38}, e_{43})$; $e_{45} = F_8(e_{44}, e_{44})$;
return e_{45}

Obrázek A.6: Nejlepší klasifikátor pro třídu 5

def $class6(p)$:

$e_0 = F_9(p_{78}, p_{45})$; $e_1 = F_5(p_{47}, p_{49})$; $e_2 = F_5(p_{148}, p_{131})$; $e_3 = F_6(p_{66}, p_{65})$;
$e_4 = F_6(e_0, e_3)$; $e_5 = F_9(p_{147}, p_{161})$; $e_6 = F_9(p_{117}, p_{115})$; $e_7 = F_4(p_{118})$;
$e_8 = F_6(p_{67}, e_4)$; $e_9 = F_9(p_{146}, e_2)$; $e_{10} = F_8(p_{31}, p_{95})$; $e_{11} = F_9(p_{116}, e_6)$;
$e_{12} = F_8(p_{23}, p_{63})$; $e_{13} = F_5(e_5, e_9)$; $e_{14} = F_9(p_{158}, p_{52})$; $e_{15} = F_9(e_{14}, e_1)$;
$e_{16} = F_6(e_8, e_{15})$; $e_{17} = F_5(e_{13}, e_{11})$; $e_{18} = F_{11}(p_{55}, e_{16})$; $e_{19} = F_8(e_{18}, e_{17})$;
$e_{20} = F_5(p_{19}, p_{78})$; $e_{21} = F_{11}(e_{10}, p_{12})$; $e_{22} = F_{10}(p_{97}, p_{141})$; $e_{23} = F_5(p_{34}, p_{64})$;
$e_{24} = F_{10}(e_{12}, e_7)$; $e_{25} = F_{11}(e_{20}, e_{19})$; $e_{26} = F_2(e_{21})$; $e_{27} = F_9(e_{23}, p_{174})$;
$e_{28} = F_7(e_{22}, e_{22})$; $e_{29} = F_7(e_{27}, e_{25})$; $e_{30} = F_{11}(e_{24}, e_{26})$; $e_{31} = F_{11}(e_{29}, e_{28})$;
$e_{32} = F_7(e_{25}, e_{31})$; $e_{33} = F_6(e_{32}, p_{72})$; $e_{34} = F_{10}(e_{33}, e_{30})$;
return e_{34}

Obrázek A.7: Nejlepší klasifikátor pro třídu 6

def $class7(p)$:

$e_0 = F_9(p_{163}, p_{176})$; $e_1 = F_9(p_{165}, p_{91})$; $e_2 = F_9(p_{49}, p_{33})$; $e_3 = F_9(p_{59}, e_1)$;
$e_4 = F_6(p_{150}, p_{101})$; $e_5 = F_5(p_{49}, e_0)$; $e_6 = F_9(p_{50}, p_{51})$; $e_7 = F_6(e_5, p_{103})$;
$e_8 = F_5(p_{78}, p_{51})$; $e_9 = F_{10}(e_3, p_{75})$; $e_{10} = F_6(p_{118}, p_{131})$; $e_{11} = F_9(p_{48}, e_6)$;
$e_{12} = F_9(e_{10}, e_5)$; $e_{13} = F_6(p_{104}, p_{138})$; $e_{14} = F_9(p_{102}, p_{115})$; $e_{15} = F_7(p_{72}, p_{109})$;
$e_{16} = F_9(e_{13}, p_{151})$; $e_{17} = F_7(p_{37}, e_{16})$; $e_{18} = F_8(e_9, p_{175})$; $e_{19} = F_9(e_{14}, e_4)$;
$e_{20} = F_{10}(p_{193}, p_{173})$; $e_{21} = F_9(p_{34}, p_{144})$; $e_{22} = F_7(e_{11}, e_2)$; $e_{23} = F_9(e_{19}, e_7)$;
$e_{24} = F_2(p_{169})$; $e_{25} = F_5(e_{22}, e_1)$; $e_{26} = F_{11}(p_{190}, e_{18})$; $e_{27} = F_{10}(p_{146}, p_9)$;
$e_{28} = F_5(p_{172}, e_8)$; $e_{29} = F_9(e_{21}, e_{25})$; $e_{30} = F_8(e_{27}, p_{189})$; $e_{31} = F_6(e_{23}, e_{28})$;
$e_{32} = F_6(e_4, e_{26})$; $e_{33} = F_{11}(e_{32}, e_{20})$; $e_{34} = F_5(e_{12}, e_{31})$; $e_{35} = F_9(e_{29}, e_{17})$;
$e_{36} = F_9(e_{35}, e_{34})$; $e_{37} = F_9(e_{36}, p_{143})$; $e_{38} = F_{11}(e_{37}, p_{170})$; $e_{39} = F_3(e_{38})$;
$e_{40} = F_9(e_{33}, p_{187})$; $e_{41} = F_5(e_{40}, e_{15})$; $e_{42} = F_9(e_{30}, e_{41})$; $e_{43} = F_7(e_{39}, e_{42})$;
$e_{44} = F_{10}(e_{43}, e_{24})$; $e_{45} = F_8(e_{44}, e_{44})$;
return e_{45}

Obrázek A.8: Nejlepší klasifikátor pro třídu 7

def $class8(p)$:

$e_0 = F_{11}(p_{131}, p_{86})$; $e_1 = F_9(p_{94}, p_{46})$; $e_2 = F_6(p_{175}, p_{161})$; $e_3 = F_4(p_{104})$;
$e_4 = F_{11}(e_1, p_{79})$; $e_5 = F_9(e_2, p_{160})$; $e_6 = F_6(e_4, p_{115})$; $e_7 = F_5(p_{63}, p_{122})$;
$e_8 = F_5(p_{105}, e_5)$; $e_9 = F_6(p_{76}, p_{89})$; $e_{10} = F_1(p_2)$; $e_{11} = F_9(p_{123}, e_6)$;
$e_{12} = F_8(e_8, e_3)$; $e_{13} = F_8(e_2, p_{132})$; $e_{14} = F_8(e_{13}, p_{95})$; $e_{15} = F_7(p_{102}, p_{151})$;
$e_{16} = F_{11}(e_{13}, p_{130})$; $e_{17} = F_{11}(p_5, e_{10})$; $e_{18} = F_6(e_{15}, e_7)$; $e_{19} = F_5(e_9, e_{12})$;
$e_{20} = F_{11}(e_{14}, e_{19})$; $e_{21} = F_9(e_{16}, e_0)$; $e_{22} = F_6(p_{78}, p_{135})$; $e_{23} = F_6(e_{21}, p_{168})$;
$e_{24} = F_6(e_{23}, e_{20})$; $e_{25} = F_8(p_{147}, e_{22})$; $e_{26} = F_9(e_{18}, e_{25})$; $e_{27} = F_9(e_{26}, e_{24})$;
$e_{28} = F_6(e_{27}, e_{11})$; $e_{29} = F_{10}(e_{28}, e_{17})$; $e_{30} = F_8(e_{29}, e_{29})$;
return e_{30}

Obrázek A.9: Nejlepší klasifikátor pro třídu 8

def $class9(p)$:

$e_0 = F_9(p_{90}, p_4)$; $e_1 = F_6(p_{54}, p_{58})$; $e_2 = F_7(p_{109}, p_{53})$; $e_3 = F_5(p_{117}, e_0)$;
$e_4 = F_2(p_{104})$; $e_5 = F_9(e_3, e_2)$; $e_6 = F_{11}(p_{158}, p_{145})$; $e_7 = F_5(p_{92}, p_{89})$;
$e_8 = F_6(p_{88}, p_{101})$; $e_9 = F_{11}(p_{144}, e_1)$; $e_{10} = F_6(p_{117}, p_{130})$; $e_{11} = F_9(e_7, e_8)$;
$e_{12} = F_7(p_{157}, p_{137})$; $e_{13} = F_9(e_4, p_{77})$; $e_{14} = F_{11}(p_{35}, e_5)$; $e_{15} = F_{11}(p_{68}, p_{84})$;
$e_{16} = F_5(p_{162}, p_{160})$; $e_{17} = F_{11}(e_{16}, e_{12})$; $e_{18} = F_6(p_{161}, p_{175})$; $e_{19} = F_9(e_{10}, e_0)$;
$e_{20} = F_9(e_{11}, e_6)$; $e_{21} = F_5(p_{163}, e_{18})$; $e_{22} = F_{11}(e_{20}, p_{63})$; $e_{23} = F_{11}(e_{13}, e_{19})$;
$e_{24} = F_6(e_{14}, p_{95})$; $e_{25} = F_{11}(e_{21}, p_{123})$; $e_{26} = F_8(e_{20}, e_{17})$; $e_{27} = F_6(p_{49}, e_{22})$;
$e_{28} = F_5(e_{24}, e_9)$; $e_{29} = F_8(e_{26}, e_{28})$; $e_{30} = F_8(e_{23}, e_{15})$; $e_{31} = F_5(e_{30}, e_{27})$;
$e_{32} = F_{10}(e_{31}, e_{25})$; $e_{33} = F_{10}(e_{29}, e_{32})$; $e_{34} = F_{10}(e_{33}, p_{123})$; $e_{35} = F_{11}(e_{34}, e_{33})$;
return e_{35}

Obrázek A.10: Nejlepší klasifikátor pro třídu 9